
pip-tools

Release 6.12.4.dev0+g4219171.d20230301

pip-tools Contributors

Mar 01, 2023

CONTENTS:

1	pip-tools = pip-compile + pip-sync	3
1.1	Installation	3
1.2	Example usage for pip-compile	3
1.3	Example usage for pip-sync	10
1.4	Should I commit requirements.in and requirements.txt to source control?	10
1.5	Cross-environment usage of requirements.in/requirements.txt and pip-compile	11
1.6	Other useful tools	11
1.7	Deprecations	11
1.8	A Note on Resolvers	11
1.9	Versions and compatibility	12
1.10	Indices and tables	31

PIP-TOOLS = PIP-COMPILE + PIP-SYNC

A set of command line tools to help you keep your `pip`-based packages fresh, even when you've pinned them. You do pin them, right? (In building your Python application and its dependencies for production, you want to make sure that your builds are predictable and deterministic.)

1.1 Installation

Similar to `pip`, `pip-tools` must be installed in each of your project's [virtual environments](#):

```
$ source /path/to/venv/bin/activate
(venv) $ python -m pip install pip-tools
```

Note: all of the remaining example commands assume you've activated your project's virtual environment.

1.2 Example usage for `pip-compile`

The `pip-compile` command lets you compile a `requirements.txt` file from your dependencies, specified in either `pyproject.toml`, `setup.cfg`, `setup.py`, or `requirements.in`.

Run it with `pip-compile` or `python -m piptools compile`. If you use multiple Python versions, you can also run `py -X.Y -m piptools compile` on Windows and `pythonX.Y -m piptools compile` on other systems.

`pip-compile` should be run from the same virtual environment as your project so conditional dependencies that require a specific Python version, or other environment markers, resolve relative to your project's environment.

Note: If `pip-compile` finds an existing `requirements.txt` file that fulfils the dependencies then no changes will be made, even if updates are available. To compile from scratch, first delete the existing `requirements.txt` file, or see [Updating requirements](#) for alternative approaches.

1.2.1 Requirements from pyproject.toml

The `pyproject.toml` file is the [latest standard](#) for configuring packages and applications, and is recommended for new projects. `pip-compile` supports both installing your `project.dependencies` as well as your `project.optional-dependencies`. Thanks to the fact that this is an official standard, you can use `pip-compile` to pin the dependencies in projects that use modern standards-adhering packaging tools like [Hatch](#) or [flit](#).

Suppose you have a Django application that is packaged using [Hatch](#), and you want to pin it for production. You also want to pin your development tools in a separate pin file. You declare `django` as a dependency and create an optional dependency `dev` that includes `pytest`:

```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "my-cool-django-app"
version = "42"
dependencies = ["django"]

[project.optional-dependencies]
dev = ["pytest"]
```

You can produce your pin files as easily as:

```
$ pip-compile -o requirements.txt pyproject.toml
#
# This file is autogenerated by pip-compile with python 3.10
# To update, run:
#
#   pip-compile --output-file=requirements.txt pyproject.toml
#
asgiref==3.5.2
  # via django
django==4.1
  # via my-cool-django-app (pyproject.toml)
sqlparse==0.4.2
  # via django

$ pip-compile --extra dev -o dev-requirements.txt pyproject.toml
#
# This file is autogenerated by pip-compile with python 3.10
# To update, run:
#
#   pip-compile --extra=dev --output-file=dev-requirements.txt pyproject.toml
#
asgiref==3.5.2
  # via django
attrs==22.1.0
  # via pytest
django==4.1
  # via my-cool-django-app (pyproject.toml)
```

(continues on next page)

(continued from previous page)

```

iniconfig==1.1.1
    # via pytest
packaging==21.3
    # via pytest
pluggy==1.0.0
    # via pytest
py==1.11.0
    # via pytest
pyparsing==3.0.9
    # via packaging
pytest==7.1.2
    # via my-cool-django-app (pyproject.toml)
sqlparse==0.4.2
    # via django
tomli==2.0.1
    # via pytest

```

This is great for both pinning your applications, but also to keep the CI of your open-source Python package stable.

1.2.2 Requirements from `setup.py` and `setup.cfg`

`pip-compile` has also full support for `setup.py`- and `setup.cfg`-based projects that use `setuptools`.

Just define your dependencies and extras as usual and run `pip-compile` as above.

1.2.3 Requirements from `requirements.in`

You can also use plain text files for your requirements (e.g. if you don't want your application to be a package). To use a `requirements.in` file to declare the Django dependency:

```

# requirements.in
django

```

Now, run `pip-compile requirements.in`:

```

$ pip-compile requirements.in
#
# This file is autogenerated by pip-compile
# To update, run:
#
#   pip-compile requirements.in
#
asgiref==3.2.3
    # via django
django==3.0.3
    # via -r requirements.in
pytz==2019.3
    # via django
sqlparse==0.3.0
    # via django

```

And it will produce your `requirements.txt`, with all the Django dependencies (and all underlying dependencies) pinned.

1.2.4 Updating requirements

`pip-compile` generates a `requirements.txt` file using the latest versions that fulfil the dependencies you specify in the supported files.

If `pip-compile` finds an existing `requirements.txt` file that fulfils the dependencies then no changes will be made, even if updates are available.

To force `pip-compile` to update all packages in an existing `requirements.txt`, run `pip-compile --upgrade`.

To update a specific package to the latest or a specific version use the `--upgrade-package` or `-P` flag:

```
# only update the django package
$ pip-compile --upgrade-package django

# update both the django and requests packages
$ pip-compile --upgrade-package django --upgrade-package requests

# update the django package to the latest, and requests to v2.0.0
$ pip-compile --upgrade-package django --upgrade-package requests==2.0.0
```

You can combine `--upgrade` and `--upgrade-package` in one command, to provide constraints on the allowed upgrades. For example to upgrade all packages whilst constraining requests to the latest version less than 3.0:

```
$ pip-compile --upgrade --upgrade-package 'requests<3.0'
```

1.2.5 Using hashes

If you would like to use *Hash-Checking Mode* available in `pip` since version 8.0, `pip-compile` offers `--generate-hashes` flag:

```
$ pip-compile --generate-hashes requirements.in
#
# This file is autogenerated by pip-compile
# To update, run:
#
#   pip-compile --generate-hashes requirements.in
#
asgiref==3.2.3 \
--hash=sha256:7e06d934a7718bf3975acbf87780ba678957b87c7adc056f13b6215d610695a0 \
--hash=sha256:ea448f92fc35a0ef4b1508f53a04c4670255a3f33d22a81c8fc9c872036adbe5 \
# via django
django==3.0.3 \
--hash=sha256:2f1ba1db8648484dd5c238fb62504777b7ad090c81c5f1fd8d5eb5ec21b5f283 \
--hash=sha256:c91c91a7ad6ef67a874a4f76f58ba534f9208412692a840e1d125eb5c279cb0a \
# via -r requirements.in
pytz==2019.3 \
--hash=sha256:1c557d7d0e871de1f5ccd5833f60fb2550652da6be2693c1e02300743d21500d \
--hash=sha256:b02c06db6cf09c12dd25137e563b31700d3b80fcc4ad23abb7a315f2789819be \
# via django
```

(continues on next page)

(continued from previous page)

```
sqlparse==0.3.0 \
  --hash=sha256:40afe6b8d4b1117e7dff5504d7a8ce07d9a1b15aeade8a2d10f130a834f8177 \
  --hash=sha256:7c3dca29c022744e95b547e867cee89f4fce4373f3549ccd8797d8eb52cdb873 \
  # via django
```

1.2.6 Output File

To output the pinned requirements in a filename other than `requirements.txt`, use `--output-file`. This might be useful for compiling multiple files, for example with different constraints on `django` to test a library with both versions using `tox`:

```
$ pip-compile --upgrade-package 'django<1.0' --output-file requirements-django0x.txt
$ pip-compile --upgrade-package 'django<2.0' --output-file requirements-django1x.txt
```

Or to output to standard output, use `--output-file=-`:

```
$ pip-compile --output-file=- > requirements.txt
$ pip-compile - --output-file=- < requirements.in > requirements.txt
```

1.2.7 Forwarding options to pip

Any valid `pip` flags or arguments may be passed on with `pip-compile`'s `--pip-args` option, e.g.

```
$ pip-compile requirements.in --pip-args "--retries 10 --timeout 30"
```

1.2.8 Configuration

You might be wrapping the `pip-compile` command in another script. To avoid confusing consumers of your custom script you can override the update command generated at the top of requirements files by setting the `CUSTOM_COMPILE_COMMAND` environment variable.

```
$ CUSTOM_COMPILE_COMMAND="./pipcompilewrapper" pip-compile requirements.in
#
# This file is autogenerated by pip-compile
# To update, run:
#
#   ./pipcompilewrapper
#
asgiref==3.2.3
  # via django
django==3.0.3
  # via -r requirements.in
pytz==2019.3
  # via django
sqlparse==0.3.0
  # via django
```

1.2.9 Workflow for layered requirements

If you have different environments that you need to install different but compatible packages for, then you can create layered requirements files and use one layer to constrain the other.

For example, if you have a Django project where you want the newest 2.1 release in production and when developing you want to use the Django debug toolbar, then you can create two *.in files, one for each layer:

```
# requirements.in
django<2.2
```

At the top of the development requirements dev-requirements.in you use -c requirements.txt to constrain the dev requirements to packages already selected for production in requirements.txt.

```
# dev-requirements.in
-c requirements.txt
django-debug-toolbar
```

First, compile requirements.txt as usual:

```
$ pip-compile
#
# This file is autogenerated by pip-compile
# To update, run:
#
#   pip-compile
#
django==2.1.15
    # via -r requirements.in
pytz==2019.3
    # via django
```

Now compile the dev requirements and the requirements.txt file is used as a constraint:

```
$ pip-compile dev-requirements.in
#
# This file is autogenerated by pip-compile
# To update, run:
#
#   pip-compile dev-requirements.in
#
django-debug-toolbar==2.2
    # via -r dev-requirements.in
django==2.1.15
    # via
    #   -c requirements.txt
    #   django-debug-toolbar
pytz==2019.3
    # via
    #   -c requirements.txt
    #   django
sqlparse==0.3.0
    # via django-debug-toolbar
```

As you can see above, even though a 2.2 release of Django is available, the dev requirements only include a 2.1

version of Django because they were constrained. Now both compiled requirements files can be installed safely in the dev environment.

To install requirements in production stage use:

```
$ pip-sync
```

You can install requirements in development stage by:

```
$ pip-sync requirements.txt dev-requirements.txt
```

1.2.10 Version control integration

You might use `pip-compile` as a hook for the `pre-commit`. See [pre-commit docs](#) for instructions. Sample `.pre-commit-config.yaml`:

```
repos:
- repo: https://github.com/jazzband/pip-tools
  rev: 6.12.3
  hooks:
  - id: pip-compile
```

You might want to customize `pip-compile` args by configuring `args` and/or `files`, for example:

```
repos:
- repo: https://github.com/jazzband/pip-tools
  rev: 6.12.3
  hooks:
  - id: pip-compile
    files: ^requirements/production\.(in|txt)$
    args: [--index-url=https://example.com, requirements/production.in]
```

If you have multiple requirement files make sure you create a hook for each file.

```
repos:
- repo: https://github.com/jazzband/pip-tools
  rev: 6.12.3
  hooks:
  - id: pip-compile
    name: pip-compile setup.py
    files: ^(setup\.py|requirements\.txt)$
  - id: pip-compile
    name: pip-compile requirements-dev.in
    args: [requirements-dev.in]
    files: ^requirements-dev\.(in|txt)$
  - id: pip-compile
    name: pip-compile requirements-lint.in
    args: [requirements-lint.in]
    files: ^requirements-lint\.(in|txt)$
  - id: pip-compile
    name: pip-compile requirements.txt
    args: [requirements.txt]
    files: ^requirements\.(in|txt)$
```

1.3 Example usage for pip-sync

Now that you have a `requirements.txt`, you can use `pip-sync` to update your virtual environment to reflect exactly what's in there. This will install/upgrade/uninstall everything necessary to match the `requirements.txt` contents.

Run it with `pip-sync` or `python -m piptools sync`. If you use multiple Python versions, you can also run `py -X.Y -m piptools sync` on Windows and `pythonX.Y -m piptools sync` on other systems.

`pip-sync` must be installed into and run from the same virtual environment as your project to identify which packages to install or upgrade.

Be careful: `pip-sync` is meant to be used only with a `requirements.txt` generated by `pip-compile`.

```
$ pip-sync
Uninstalling flake8-2.4.1:
  Successfully uninstalled flake8-2.4.1
Collecting click==4.1
  Downloading click-4.1-py2.py3-none-any.whl (62kB)
    100% |.....| 65kB 1.8MB/s
  Found existing installation: click 4.0
    Uninstalling click-4.0:
      Successfully uninstalled click-4.0
Successfully installed click-4.1
```

To sync multiple `*.txt` dependency lists, just pass them in via command line arguments, e.g.

```
$ pip-sync dev-requirements.txt requirements.txt
```

Passing in empty arguments would cause it to default to `requirements.txt`.

Any valid `pip` install flags or arguments may be passed with `pip-sync`'s `--pip-args` option, e.g.

```
$ pip-sync requirements.txt --pip-args "--no-cache-dir --no-deps"
```

Note: `pip-sync` will not upgrade or uninstall packaging tools like `setuptools`, `pip`, or `pip-tools` itself. Use `python -m pip install --upgrade` to upgrade those packages.

1.4 Should I commit requirements.in and requirements.txt to source control?

Generally, yes. If you want a reproducible environment installation available from your source control, then yes, you should commit both `requirements.in` and `requirements.txt` to source control.

Note that if you are deploying on multiple Python environments (read the section below), then you must commit a separate output file for each Python environment. We suggest to use the `{env}-requirements.txt` format (ex: `win32-py3.7-requirements.txt`, `macos-py3.10-requirements.txt`, etc.).

1.5 Cross-environment usage of requirements.in/requirements.txt and pip-compile

The dependencies of a package can change depending on the Python environment in which it is installed. Here, we define a Python environment as the combination of Operating System, Python version (3.7, 3.8, etc.), and Python implementation (CPython, PyPy, etc.). For an exact definition, refer to the possible combinations of [PEP 508 environment markers](#).

As the resulting `requirements.txt` can differ for each environment, users must execute `pip-compile` **on each Python environment separately** to generate a `requirements.txt` valid for each said environment. The same `requirements.in` can be used as the source file for all environments, using [PEP 508 environment markers](#) as needed, the same way it would be done for regular `pip` cross-environment usage.

If the generated `requirements.txt` remains exactly the same for all Python environments, then it can be used across Python environments safely. **But** users should be careful as any package update can introduce environment-dependant dependencies, making any newly generated `requirements.txt` environment-dependant too. As a general rule, it's advised that users should still always execute `pip-compile` on each targeted Python environment to avoid issues.

1.6 Other useful tools

- `pipdeptree` to print the dependency tree of the installed packages.
- `requirements.in/requirements.txt` syntax highlighting:
 - `requirements.txt.vim` for Vim.
 - Python extension for VS Code for VS Code.
 - `pip-requirements.el` for Emacs.

1.7 Deprecations

This section lists `pip-tools` features that are currently deprecated.

- In future versions, the `--allow-unsafe` behavior will be enabled by default. Use `--no-allow-unsafe` to keep the old behavior. It is recommended to pass the `--allow-unsafe` now to adapt to the upcoming change.
- Legacy resolver is deprecated and will be removed in future versions. Use `--resolver=backtracking` instead.

1.8 A Note on Resolvers

You can choose from either the legacy or the backtracking resolver. The backtracking resolver is recommended, and will become the default with the 7.0 release.

Use it now with the `--resolver=backtracking` option to `pip-compile`.

The legacy resolver will occasionally fail to resolve dependencies. The backtracking resolver is more robust, but can take longer to run in general.

You can continue using the legacy resolver with `--resolver=legacy`.

1.9 Versions and compatibility

The table below summarizes the latest `pip-tools` versions with the required `pip` and Python versions. Generally, `pip-tools` supports the same Python versions as the required `pip` versions.

pip-tools	pip	Python
4.5.*	8.1.3 - 20.0.2	2.7, 3.5 - 3.8
5.0.0 - 5.3.0	20.0 - 20.1.1	2.7, 3.5 - 3.8
5.4.0	20.1 - 20.3.*	2.7, 3.5 - 3.8
5.5.0	20.1 - 20.3.*	2.7, 3.5 - 3.9
6.0.0 - 6.3.1	20.3 - 21.2.*	3.6 - 3.9
6.4.0	21.2 - 21.3.*	3.6 - 3.10
6.5.0 - 6.10.0	21.2 - 22.3.*	3.7 - 3.11
6.11.0+	22.2+	3.7 - 3.11

1.9.1 Contributing

This is a [Jazzband](#) project. By contributing you agree to abide by the [Contributor Code of Conduct](#) and follow the [guidelines](#).

Project Contribution Guidelines

Here are a few additional or emphasized guidelines to follow when contributing to `pip-tools`:

- If you need to have a virtualenv outside of `tox`, it is possible to reuse its configuration to provision it with `tox devenv`.
- Always provide tests for your changes and run `tox -p all` to make sure they are passing the checks locally.
- Give a clear one-line description in the PR (that the maintainers can add to [CHANGELOG](#) afterwards).
- Wait for the review of at least one other contributor before merging (even if you're a Jazzband member).
- Before merging, assign the PR to a milestone for a version to help with the release process.

The only exception to those guidelines is for trivial changes, such as documentation corrections or contributions that do not change `pip-tools` itself.

Contributions following these guidelines are always welcomed, encouraged and appreciated.

Project Release Process

Jazzband aims to give full access to all members, including performing releases, as described in the [Jazzband Releases documentation](#).

To help keeping track of the releases and their changes, here's the current release process:

- Check to see if any recently merged PRs are missing from the milestone of the version about to be released.
- Create a branch for the release. *Ex: release-3.4.0*.
- Update the [CHANGELOG](#) with the version, date and add the text from `drafter release`.
- Push the branch to your fork and create a pull request.

- Merge the pull request after the changes being approved.
- Make sure that the tests/CI still pass.
- Once ready, go to [releases](#) page and publish the latest draft release. This will push a tag on the HEAD of the main branch, trigger the CI pipeline and deploy a pip-tools release in the **Jazzband private package index** upon success.
- The pip-tools “lead” project members will receive an email notification to review the release and deploy it to the public PyPI if all is correct.
- Once the release to the public PyPI is confirmed, close the milestone.

Please be mindful of other before and when performing a release, and use this access responsibly.

Do not hesitate to ask questions if you have any before performing a release.

1.9.2 Changelog

6.12.3 (2022-03-01)

Bug Fixes:

- Remove extras from user-supplied constraints in backtracking resolver (#1808). Thanks @thomdixon
- Fix for sync error when the ireqs being merged have no names (#1802). Thanks @richafrank

6.12.2 (2022-12-25)

Bug Fixes:

- Raise error if input and output filenames are matched (#1787). Thanks @atugushev
- Add `pyproject.toml` as default input file format (#1780). Thanks @berislavlopac
- Fix a regression with unsafe packages for `--allow-unsafe` (#1788). Thanks @q0w

6.12.1 (2022-12-16)

Bug Fixes:

- Set explicitly packages for `setuptools` (#1782). Thanks @q0w

6.12.0 (2022-12-13)

Features:

- Add `--no-index` flag to `pip-compile` (#1745). Thanks @atugushev

Bug Fixes:

- Treat `--upgrade-packages` PKGSPECs as constraints (not just minimums), consistently (#1578). Thanks @AndydeCleyre
- Filter out the user provided unsafe packages (#1766). Thanks @q0w
- Adopt PEP-621 for packaging (#1763). Thanks @ssbarnea

6.11.0 (2022-11-30)

Features:

- Add `pyproject.toml` file (#1643). Thanks @otherJL0
- Support build isolation using `setuptools/pyproject.toml` requirement files (#1727). Thanks @atugushev

Bug Fixes:

- Improve punctuation/grammar with `pip-compile` header (#1547). Thanks @blueyed
- Generate hashes for all available candidates (#1723). Thanks @neykov

Other Changes:

- Bump click minimum version to `>= 8` (#1733). Thanks @atugushev
- Bump pip minimum version to `>= 22.2` (#1729). Thanks @atugushev

6.10.0 (2022-11-13)

Features:

- Deprecate `pip-compile --resolver=legacy` (#1724). Thanks @atugushev
- Prompt user to use the backtracking resolver on errors (#1719). Thanks @maxfenv
- Add support for Python 3.11 final (#1708). Thanks @hugovk
- Add `--newline=[LF|CRLF|native|preserve]` option to `pip-compile` (#1652). Thanks @AndydeCleyre

Bug Fixes:

- Fix inconsistent handling of constraints comments with backtracking resolver (#1713). Thanks @mknieuwallner
- Fix some encoding warnings in Python 3.10 (PEP 597) (#1614). Thanks @GalaxySnail

Other Changes:

- Update pip-tools version in the README's pre-commit examples (#1701). Thanks @Kludex
- Document use of the backtracking resolver (#1718). Thanks @maxfenv
- Use HTTPS in a readme link (#1716). Thanks @Arhell

6.9.0 (2022-10-05)

Features:

- Add `--all-extras` flag to `pip-compile` (#1630). Thanks @apljungquist
- Support Exclude Package with custom unsafe packages (#1509). Thanks @hmc-cs-mdrissi

Bug Fixes:

- Fix compile cached vcs packages (#1649). Thanks @atugushev
- Include `py.typed` in wheel file (#1648). Thanks @FlorentJeannot

Other Changes:

- Add `pyproject.toml` & modern packaging to introduction. (#1668). Thanks @hynek

6.8.0 (2022-06-30)

Features:

- Add support for pip's 2020 dependency resolver. Use `pip-compile --resolver backtracking` to enable new resolver (#1539). Thanks @atugushev

6.7.0 (2022-06-27)

Features:

- Support for the `importlib.metadata` metadata implementation (#1632). Thanks @richafrank

Bug Fixes:

- Instantiate a new accumulator `InstallRequirement` for `combine_install_requirements` output (#1519). Thanks @richafrank

Other Changes:

- Replace direct usage of the `pep517` module with the `build` module, for loading project metadata (#1629). Thanks @AndydeCleyre

6.6.2 (2022-05-23)

Bug Fixes:

- Update `PyPIRepository::resolve_reqs()` for `pip>=22.1.1` (#1624). Thanks @m000

6.6.1 (2022-05-13)

Bug Fixes:

- Fix support for `pip>=22.1` (#1618). Thanks @wizpig64

6.6.0 (2022-04-06)

Features:

- Add support for `pip>=22.1` (#1607). Thanks @atugushev

Bug Fixes:

- Ensure `pip-compile --dry-run --quiet` still shows what would be done, while omitting the dry run message (#1592). Thanks @AndydeCleyre
- Fix `--generate-hashes` when hashes are computed from files (#1540). Thanks @RazerM

6.5.1 (2022-02-08)

Bug Fixes:

- Ensure canonicalized requirement names are used as keys, to prevent unnecessary reinstallations during sync (#1572). Thanks @AndydeCleyre

6.5.0 (2022-02-04)

Features:

- Add support for pip>=22.0, drop support for Python 3.6 (#1567). Thanks @di
- Test on Python 3.11 (#1527). Thanks @hugovk

Other Changes:

- Minor doc edits (#1445). Thanks @ssiano

6.4.0 (2021-10-12)

Features:

- Add support for pip>=21.3 (#1501). Thanks @atugushev
- Add support for Python 3.10 (#1497). Thanks @joshuadavidthomas

Other Changes:

- Bump pip minimum version to >= 21.2 (#1500). Thanks @atugushev

6.3.1 (2021-10-08)

Bug Fixes:

- Ensure pip-tools unions dependencies of multiple declarations of a package with different extras (#1486). Thanks @richafrank
- Allow comma-separated arguments for --extra (#1493). Thanks @AndydeCleyre
- Improve clarity of help text for options supporting multiple (#1492). Thanks @AndydeCleyre

6.3.0 (2021-09-21)

Features:

- Enable single-line annotations with pip-compile --annotation-style=line (#1477). Thanks @AndydeCleyre
- Generate PEP 440 direct reference whenever possible (#1455). Thanks @FlorentJeannot
- PEP 440 Direct Reference support (#1392). Thanks @FlorentJeannot

Bug Fixes:

- Change log level of hash message (#1460). Thanks @plannigan
- Allow passing --no-upgrade option (#1438). Thanks @ssbarnea

6.2.0 (2021-06-22)

Features:

- Add `--emit-options/--no-emit-options` flags to `pip-compile` (#1123). Thanks @atugushev
- Add `--python-executable` option for `pip-sync` (#1333). Thanks @MaratFM
- Log which python version was used during compile (#828). Thanks @graingert

Bug Fixes:

- Fix `pip-compile` package ordering (#1419). Thanks @adamsol
- Add `--strip-extras` option to `pip-compile` for producing constraint compatible output (#1404). Thanks @ssbarnea
- Fix `click v7 version_option` compatibility (#1410). Thanks @FuegoFro
- Pass `package_name` explicitly in `click.version_option` decorators for compatibility with `click>=8.0` (#1400). Thanks @nicoa

Other Changes:

- Document updating requirements with `pre-commit` hooks (#1387). Thanks @microcat49
- Add `setuptools` and `wheel` dependencies to the `setup.cfg` (#889). Thanks @jayvdb
- Improve instructions for new contributors (#1394). Thanks @FlorentJeannot
- Better explain role of existing `requirements.txt` (#1369). Thanks @mikepqr

6.1.0 (2021-04-14)

Features:

- Add support for `pyproject.toml` or `setup.cfg` as input dependency file (PEP-517) for `pip-compile` (#1356). Thanks @orsinium
- Add `pip-compile --extra` option to specify `extras_require` dependencies (#1363). Thanks @orsinium

Bug Fixes:

- Restore ability to set compile cache with env var `PIP_TOOLS_CACHE_DIR` (#1368). Thanks @AndydeCleyre

6.0.1 (2021-03-15)

Bug Fixes:

- Fixed a bug with undeclared dependency on `importlib-metadata` at Python 3.6 (#1353). Thanks @atugushev

Dependencies:

- Add `pep517` dependency (#1353). Thanks @atugushev

6.0.0 (2021-03-12)

Backwards Incompatible Changes:

- Remove support for EOL Python 3.5 and 2.7 (#1243). Thanks @jdufresne
- Remove deprecated `--index/--no-index` option from `pip-compile` (#1234). Thanks @jdufresne

Features:

- Use `pep517` to parse dependencies metadata from `setup.py` (#1311). Thanks @astrojuanlu

Bug Fixes:

- Fix a bug where `pip-compile` with `setup.py` would not include dependencies with environment markers (#1311). Thanks @astrojuanlu
- Prefer `===` over `==` when generating `requirements.txt` if a dependency was pinned with `===` (#1323). Thanks @IceTDrinker
- Fix a bug where `pip-compile` with `setup.py` in nested folder would generate `setup.txt` output file (#1324). Thanks @peymanslh
- Write out default index when it is provided as `--extra-index-url` (#1325). Thanks @fahrradflucht

Dependencies:

- Bump `pip` minimum version to `>= 20.3` (#1340). Thanks @atugushev

5.5.0 (2020-12-31)

Features:

- Add Python 3.9 support (1222). Thanks @jdufresne
- Improve formatting of long “via” annotations (1237). Thanks @jdufresne
- Add `--verbose` and `--quiet` options to `pip-sync` (1241). Thanks @jdufresne
- Add `--no-allow-unsafe` option to `pip-compile` (1265). Thanks @jdufresne

Bug Fixes:

- Restore `PIP_EXISTS_ACTION` environment variable to its previous state when resolve dependencies in `pip-compile` (1255). Thanks @jdufresne

Dependencies:

- Remove `six` dependency in favor `pip`'s vendored `six` (1240). Thanks @jdufresne

Improved Documentation:

- Add `pip-requirements.el` (for Emacs) to useful tools to `README` (#1244). Thanks @jdufresne
- Add supported Python versions to `README` (#1246). Thanks @jdufresne

5.4.0 (2020-11-21)

Features:

- Add `pip>=20.3` support (1216). Thanks @atugushev and @AndydeCleyre
- Exclude `--no-reuse-hashes` option from «command to run» header (1197). Thanks @graingert

Dependencies:

- Bump `pip` minimum version to `>= 20.1` (1191). Thanks @atugushev and @AndydeCleyre

5.3.1 (2020-07-31)

Bug Fixes:

- Fix `pip-20.2` compatibility issue that caused `pip-tools` to sometime fail to stabilize in a constant number of rounds (1194). Thanks @vphilippon

5.3.0 (2020-07-26)

Features:

- Add `-h` alias for `--help` option to `pip-sync` and `pip-compile` (1163). Thanks @jan25
- Add `pip>=20.2` support (1168). Thanks @atugushev
- `pip-sync` now exists with code 1 on `--dry-run` (1172). Thanks @francisbrito
- `pip-compile` now doesn't resolve constraints from `-c constraints.txt` that are not (yet) requirements (1175). Thanks @clsgrnc
- Add `--reuse-hashes/--no-reuse-hashes` options to `pip-compile` (1177). Thanks @graingert

5.2.1 (2020-06-09)

Bug Fixes:

- Fix a bug where `pip-compile` would lose some dependencies on update a `requirements.txt` (1159). Thanks @richafrank

5.2.0 (2020-05-27)

Features:

- Show basename of URLs when `pip-compile` generates hashes in a verbose mode (1113). Thanks @atugushev
- Add `--emit-index-url/--no-emit-index-url` options to `pip-compile` (1130). Thanks @atugushev

Bug Fixes:

- Fix a bug where `pip-compile` would ignore some of package versions when `PIP_PREFER_BINARY` is set on (1119). Thanks @atugushev
- Fix leaked URLs with credentials in the debug output of `pip-compile`. (1146). Thanks @atugushev
- Fix a bug where URL requirements would have name collisions (1149). Thanks @geokala

Deprecations:

- Deprecate `--index/--no-index` in favor of `--emit-index-url/--no-emit-index-url` options in `pip-compile` (1130). Thanks @atugushev

Other Changes:

- Switch to `setuptools` declarative syntax through `setup.cfg` (1141). Thanks @jdufresne

5.1.2 (2020-05-05)

Bug Fixes:

- Fix grouping of editables and non-editables requirements (1132). Thanks @richafrank

5.1.1 (2020-05-01)

Bug Fixes:

- Fix a bug where `pip-compile` would generate hashes for `*.egg` files (#1122). Thanks @atugushev

5.1.0 (2020-04-27)

Features:

- Show progress bar when downloading packages in `pip-compile` verbose mode (#949). Thanks @atugushev
- `pip-compile` now gets hashes from PyPI JSON API (if available) which significantly increases the speed of hashes generation (#1109). Thanks @atugushev

5.0.0 (2020-04-16)

Backwards Incompatible Changes:

- `pip-tools` now requires `pip>=20.0` (previously `8.1.x - 20.0.x`). Windows users, make sure to use `python -m pip install pip-tools` to avoid issues with `pip` self-update from now on (#1055). Thanks @atugushev
- `--build-isolation` option now set on by default for `pip-compile` (#1060). Thanks @hramezani

Features:

- Exclude requirements with non-matching markers from `pip-sync` (#927). Thanks @AndydeCleyre
- Add `pre-commit` hook for `pip-compile` (#976). Thanks @atugushev
- `pip-compile` and `pip-sync` now pass anything provided to the new `--pip-args` option on to `pip` (#1080). Thanks @AndydeCleyre
- `pip-compile` output headers are now more accurate when `--` is used to escape filenames (#1080). Thanks @AndydeCleyre
- Add `pip>=20.1` support (#1088). Thanks @atugushev

Bug Fixes:

- Fix a bug where editables that are both direct requirements and constraints wouldn't appear in `pip-compile` output (#1093). Thanks @richafrank
- `pip-compile` now sorts format controls (`--no-binary/--only-binary`) to ensure consistent results (#1098). Thanks @richafrank

Improved Documentation:

- Add cross-environment usage documentation to README (#651). Thanks @vphilippon
- Add versions compatibility table to README (#1106). Thanks @atugushev

4.5.1 (2020-02-26)

Bug Fixes:

- Strip line number annotations such as “(line XX)” from file requirements, to prevent diff noise when modifying input requirement files (#1075). Thanks @adamchainz

Improved Documentation:

- Updated README example outputs for primary requirement annotations (#1072). Thanks @richafrank

4.5.0 (2020-02-20)

Features:

- Primary requirements and VCS dependencies are now get annotated with any source `.in` files and reverse dependencies (#1058). Thanks @AndydeCleyre

Bug Fixes:

- Always use normalized path for cache directory as it is required in newer versions of pip (#1062). Thanks @kammala

Improved Documentation:

- Replace outdated link in the README with rationale for pinning (#1053). Thanks @m-aciek

4.4.1 (2020-01-31)

Bug Fixes:

- Fix a bug where `pip-compile` would keep outdated options from `requirements.txt` (#1029). Thanks @atugushev
- Fix the `No handlers could be found for logger "pip.*"` error by configuring the builtin logging module (#1035). Thanks @vphilippon
- Fix a bug where dependencies of relevant constraints may be missing from output file (#1037). Thanks @jeevb
- Upgrade the minimal version of `click` from 6.0 to 7.0 version in `setup.py` (#1039). Thanks @hramezani
- Ensure that `depcache` considers the python implementation such that (for example) `cpython3.6` does not poison the results of `pypy3.6` (#1050). Thanks @asottile

Improved Documentation:

- Make the README more imperative about installing into a project’s virtual environment to avoid confusion (#1023). Thanks @tekumara
- Add a note to the README about how to install requirements on different stages to [Workflow for layered requirements](#) section (#1044). Thanks @hramezani

4.4.0 (2020-01-21)

Features:

- Add `--cache-dir` option to `pip-compile` (#1022). Thanks @richafrank
- Add `pip>=20.0` support (#1024). Thanks @atugushev

Bug Fixes:

- Fix a bug where `pip-compile --upgrade-package` would upgrade those passed packages not already required according to the `*.in` and `*.txt` files (#1031). Thanks @AndydeCleyre

4.3.0 (2019-11-25)

Features:

- Add Python 3.8 support (#956). Thanks @hramezani
- Unpin commented out unsafe packages in `requirements.txt` (#975). Thanks @atugushev

Bug Fixes:

- Fix `pip-compile` doesn't copy `--trusted-host` from `requirements.in` to `requirements.txt` (#964). Thanks @atugushev
- Add compatibility with `pip>=20.0` (#953 and #978). Thanks @atugushev
- Fix a bug where the resolver wouldn't clean up the ephemeral wheel cache (#968). Thanks @atugushev

Improved Documentation:

- Add a note to README about `requirements.txt` file, which would possibly interfere if you're compiling from scratch (#959). Thanks @hramezani

4.2.0 (2019-10-12)

Features:

- Add `--ask` option to `pip-sync` (#913). Thanks @georgek

Bug Fixes:

- Add compatibility with `pip>=19.3` (#864, #904, #910, #912 and #915). Thanks @atugushev
- Ensure `pip-compile --no-header <blank requirements.in>` creates/overwrites `requirements.txt` (#909). Thanks @AndydeCleyre
- Fix `pip-compile --upgrade-package` removes «via» annotation (#931). Thanks @hramezani

Improved Documentation:

- Add info to README about layered requirements files and `-c` flag (#905). Thanks @jamescooke

4.1.0 (2019-08-26)

Features:

- Add `--no-emit-find-links` option to `pip-compile` (#873). Thanks @jacobtolar

Bug Fixes:

- Prevent `--dry-run` log message from being printed with `--quiet` option in `pip-compile` (#861). Thanks @ddormer
- Fix resolution of requirements from Git URLs without `-e` (#879). Thanks @andersk

4.0.0 (2019-07-25)

Backwards Incompatible Changes:

- Drop support for EOL Python 3.4 (#803). Thanks @auvipy

Bug Fixes:

- Fix `pip>=19.2` compatibility (#857). Thanks @atugushev

3.9.0 (2019-07-17)

Features:

- Print provenance information when `pip-compile` fails (#837). Thanks @jakevdp

Bug Fixes:

- Output all logging to `stderr` instead of `stdout` (#834). Thanks @georgek
- Fix output file update with `--dry-run` option in `pip-compile` (#842). Thanks @shipmints and @atugushev

3.8.0 (2019-06-06)

Features:

- Options `--upgrade` and `--upgrade-package` are no longer mutually exclusive (#831). Thanks @adamchainz

Bug Fixes:

- Fix `--generate-hashes` with bare VCS URLs (#812). Thanks @jcushman
- Fix issues with `UnicodeError` when installing `pip-tools` from source in some systems (#816). Thanks @Ab-dealiJK
- Respect `--pre` option in the input file (#822). Thanks @atugushev
- Option `--upgrade-package` now works even if the output file does not exist (#831). Thanks @adamchainz

3.7.0 (2019-05-09)

Features:

- Show progressbar on generation hashes in `pip-compile` verbose mode (#743). Thanks @atugushev
- Add options `--cert` and `--client-cert` to `pip-sync` (#798). Thanks @atugushev
- Add support for `--find-links` in `pip-compile` output (#793). Thanks @estan and @atugushev
- Normalize «command to run» in `pip-compile` headers (#800). Thanks @atugushev
- Support URLs as packages (#807). Thanks @jcushman, @nim65s and @toejough

Bug Fixes:

- Fix replacing password to asterisks in `pip-compile` (#808). Thanks @atugushev

3.6.1 (2019-04-24)

Bug Fixes:

- Fix `pip>=19.1` compatibility (#795). Thanks @atugushev

3.6.0 (2019-04-03)

Features:

- Show less output on `pip-sync` with `--quiet` option (#765). Thanks @atugushev
- Support the flag `--trusted-host` in `pip-sync` (#777). Thanks @firebirdberlin

3.5.0 (2019-03-13)

Features:

- Show default index url provided by `pip` (#735). Thanks @atugushev
- Add an option to allow enabling/disabling build isolation (#758). Thanks @atugushev

Bug Fixes:

- Fix the output file for `pip-compile` with an explicit `setup.py` as source file (#731). Thanks @atugushev
- Fix order issue with generated lock file when hashes and markers are used together (#763). Thanks @milind-shakya-sp

3.4.0 (2019-02-19)

Features:

- Add option `--quiet` to `pip-compile` (#720). Thanks @bendikro
- Emit the original command to the `pip-compile`'s header (#733). Thanks @atugushev

Bug Fixes:

- Fix `pip-sync` to use `pip` script depending on a python version (#737). Thanks @atugushev

3.3.2 (2019-01-26)

Bug Fixes:

- Fix `pip-sync` with a temporary requirement file on Windows (#723). Thanks @atugushev
- Fix `pip-sync` to prevent uninstall of `stdlib` and `dev` packages (#718). Thanks @atugushev

3.3.1 (2019-01-24)

- Re-release of 3.3.0 after fixing the deployment pipeline (#716). Thanks @atugushev

3.3.0 (2019-01-23)

(Unreleased - Deployment pipeline issue, see 3.3.1)

Features:

- Added support of `pip` 19.0 (#715). Thanks @atugushev
- Add `--allow-unsafe` to update instructions in the generated `requirements.txt` (#708). Thanks @richafrank

Bug Fixes:

- Fix `pip-sync` to check hashes (#706). Thanks @atugushev

3.2.0 (2018-12-18)

Features:

- Apply version constraints specified with package upgrade option (`-P`, `--upgrade-package`) (#694). Thanks @richafrank

3.1.0 (2018-10-05)

Features:

- Added support of `pip` 18.1 (#689). Thanks @vphilippon

3.0.0 (2018-09-24)

Major changes:

- Update `pip-tools` for native `pip` 8, 9, 10 and 18 compatibility, un-vendoring `pip` to use the user-installed `pip` (#657 and #672). Thanks to @techalchemy, @suutari, @tysonclugg and @vphilippon for contributing on this.

Features:

- Removed the dependency on the external library `first` (#676). Thanks @jdufresne

2.0.2 (2018-04-28)

Bug Fixes:

- Added clearer error reporting when skipping pre-releases (#655). Thanks @WoLpH

2.0.1 (2018-04-15)

Bug Fixes:

- Added missing package data from vendored pip, such as missing cacert.pem file. Thanks @vphilippon

2.0.0 (2018-04-15)

Major changes:

- Vendored pip 9.0.3 to keep compatibility for users with pip 10.0.0 (#644). Thanks @vphilippon

Features:

- Improved the speed of pip-compile --generate-hashes by caching the hashes from an existing output file (#641). Thanks @justicz
- Added a pip-sync --user option to restrict attention to user-local directory (#642). Thanks @jbergknoff-10e
- Removed the hard dependency on setuptools (#645). Thanks @vphilippon

Bug fixes:

- The pip environment markers on top-level requirements in the source file (requirements.in) are now properly handled and will only be processed in the right environment (#647). Thanks @JoergRittinger

1.11.0 (2017-11-30)

Features:

- Allow editable packages in requirements.in with pip-compile --generate-hashes (#524). Thanks @jdufresne
- Allow for CA bundles with pip-compile --cert (#612). Thanks @khwilson
- Improved pip-compile duration with large locally available editable requirement by skipping a copy to the cache (#583). Thanks @costypetrisor
- Slightly improved the NoCandidateFound error message on potential causes (#614). Thanks @vphilippon

Bug Fixes:

- Add -markerlib to the list of PACKAGES_TO_IGNORE of pip-sync (#613).

1.10.2 (2017-11-22)

Bug Fixes:

- Fixed bug causing dependencies from invalid wheels for the current platform to be included (#571).
- pip-sync will respect environment markers in the requirements.txt (600). Thanks @hazmat345
- Converted the ReadMe to have a nice description rendering on PyPI. Thanks @bittner

1.10.1 (2017-09-27)

Bug Fixes:

- Fixed bug breaking pip-sync on Python 3, raising TypeError: '<' not supported between instances of 'InstallRequirement' and 'InstallRequirement' (#570).

1.10.0 (2017-09-27)

Features:

- --generate-hashes now generates hashes for all wheels, not only wheels for the currently running platform (#520). Thanks @jdufresne
- Added a -q/--quiet argument to the pip-sync command to reduce log output.

Bug Fixes:

- Fixed bug where unsafe packages would get pinned in generated requirements files when --allow-unsafe was not set. (#517). Thanks @dschaller
- Fixed bug where editable PyPI dependencies would have a download_dir and be exposed to git-checkout-index, (thus losing their VCS directory) and python setup.py egg_info fails. (#385 and #538). Thanks @blueyed and @dfec
- Fixed bug where some primary dependencies were annotated with “via” info comments. (#542). Thanks @quantus
- Fixed bug where pkg-resources would be removed by pip-sync in Ubuntu. (#555). Thanks @cemsbr
- Fixed bug where the resolver would sometime not stabilize on requirements specifying extras. (#566). Thanks @vphilippon
- Fixed an unicode encoding error when distribution package contains non-ASCII file names (#567). Thanks @suutari
- Fixed package hashing doing unnecessary unpacking (#557). Thanks @suutari-ai

1.9.0 (2017-04-12)

Features:

- Added ability to read requirements from setup.py instead of just requirements.in (#418). Thanks to @tysonclugg and @majuscule.
- Added a --max-rounds argument to the pip-compile command to allow for solving large requirement sets (#472). Thanks @derek-miller.
- Exclude unsafe packages' dependencies when --allow-unsafe is not in use (#441). Thanks @jdufresne.
- Exclude irrelevant pip constraints (#471). Thanks @derek-miller.

- Allow control over emitting trusted-host to the compiled requirements. (#448). Thanks @tonyseek.
- Allow running as a Python module (#461). Thanks @AndreLouisCaron.
- Preserve environment markers in generated requirements.txt. (#460). Thanks @barrywhart.

Bug Fixes:

- Fixed the `-upgrade-package` option to respect the given package list to update (#491).
- Fixed the default output file name when the source file has no extension (#488). Thanks @vphilippon
- Fixed crash on editable requirements introduced in 1.8.2.
- Fixed duplicated `-trusted-host`, `-extra-index-url` and `-index-url` in the generated requirements.

1.8.2 (2017-03-28)

- Regression fix: editable reqs were losing their dependencies after first round (#476) Thanks @matlong
- Remove duplicate index urls in generated requirements.txt (#468) Thanks @majuscule

1.8.1 (2017-03-22)

- Recalculate secondary dependencies between rounds (#378)
- Calculated dependencies could be left with wrong candidates when toplevel requirements happen to be also pinned in sub-dependencies (#450)
- Fix duplicate entries that could happen in generated requirements.txt (#427)
- Gracefully report invalid pip version (#457)
- Fix capitalization in the generated requirements.txt, packages will always be lowercased (#452)

1.8.0 (2016-11-17)

- Adds support for upgrading individual packages with a new option `--upgrade-package`. To upgrade a *specific* package to the latest or a specific version use `--upgrade-package <pkg>`. To upgrade all packages, you can still use `pip-compile --upgrade`. (#409)
- Adds support for pinning dependencies even further by including the hashes found on PyPI at compilation time, which will be re-checked when dependencies are installed at installation time. This adds protection against packages that are tampered with. (#383)
- Improve support for extras, like `hypothesis[django]`
- Drop support for pip < 8

1.7.1 (2016-10-20)

- Add `--allow-unsafe` option (#377)

1.7.0 (2016-07-06)

- Add compatibility with pip `>= 8.1.2` (#374) Thanks so much, @jmbowman!

1.6.5 (2016-05-11)

- Add warning that pip `>= 8.1.2` is not supported until 1.7.x is out

1.6.4 (2016-05-03)

- Incorporate fix for atomic file saving behaviour on the Windows platform (see #351)

1.6.3 (2016-05-02)

- PyPI won't let me upload 1.6.2

1.6.2 (2016-05-02)

- Respect pip configuration from `pip.{ini,conf}`
- Fixes for atomic-saving of output files on Windows (see #351)

1.6.1 (2016-04-06)

Minor changes:

- `pip-sync` now supports being invoked from within and outside an activated virtualenv (see #317)
- `pip-compile`: support `-U` as a shorthand for `--upgrade`
- `pip-compile`: support pip's `--no-binary` and `--binary-only` flags

Fixes:

- Change header format of output files to mention all input files

1.6 (2016-02-05)

Major change:

- `pip-compile` will by default try to fulfill package specs by looking at a previously compiled output file first, before checking PyPI. This means `pip-compile` will only update the `requirements.txt` when it absolutely has to. To get the old behaviour (picking the latest version of all packages from PyPI), use the new `--upgrade` option.

Minor changes:

- Bugfix where `pip-compile` would lose "via" info when on pip 8 (see #313)
- Ensure cache dir exists (see #315)

1.5 (2016-01-23)

- Add support for pip ≥ 8
- Drop support for pip < 7
- Fix bug where `pip-sync` fails to uninstall packages if you're using the `--no-index` (or other) flags

1.4.5 (2016-01-20)

- Add `--no-index` flag to `pip-compile` to avoid emitting `--index-url` into the output (useful if you have configured a different index in your global `~/pip/pip.conf`, for example)
- Fix: ignore stdlib backport packages, like `argparse`, when listing which packages will be installed/uninstalled (#286)
- Fix `pip-sync` failed uninstalling packages when using `--find-links` (#298)
- Explicitly error when `pip-tools` is used with pip 8.0+ (for now)

1.4.4 (2016-01-11)

- Fix: unintended change in behaviour where packages installed by `pip-sync` could accidentally get upgraded under certain conditions, even though the `requirements.txt` would dictate otherwise (see #290)

1.4.3 (2016-01-06)

- Fix: add `--index-url` and `--extra-index-url` options to `pip-sync`
- Fix: always install using `--upgrade` flag when running `pip-sync`

1.4.2 (2015-12-13)

- Fix bug where `umask` was ignored when writing requirement files (#268)

1.4.1 (2015-12-13)

- Fix bug where successive invocations of `pip-sync` with editables kept uninstalling/installing them (fixes #270)

1.4.0 (2015-12-13)

- Add command line option `-f / --find-links`
- Add command line option `--no-index`
- Add command line alias `-n` (for `--dry-run`)
- Fix a unicode issue

1.3.0 (2015-12-08)

- Support multiple requirement files to pip-compile
- Support requirements from stdin for pip-compile
- Support `-output-file` option on pip-compile, to redirect output to a file (or stdout)

1.2.0 (2015-11-30)

- Add CHANGELOG :)
- Support pip-sync'ing editable requirements
- Support extras properly (i.e. `package[foo]` syntax)

(Anything before 1.2.0 was not recorded.)

1.10 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)