pip Documentation

Release 9.0.1

pip developers

March 22, 2018

Contents

User list | Dev list | Github | PyPI | User IRC: #pypa | Dev IRC: #pypa-dev

The PyPA recommended tool for installing Python packages.

CHAPTER 1

Quickstart

First, *install pip*.

Install a package from PyPI:

```
$ pip install SomePackage
[...]
Successfully installed SomePackage
```

Install a package that's already been downloaded from PyPI or obtained from elsewhere. This is useful if the target machine does not have a network connection:

```
$ pip install SomePackage-1.0-py2.py3-none-any.whl
[...]
Successfully installed SomePackage
```

Show what files were installed:

```
$ pip show --files SomePackage
Name: SomePackage
Version: 1.0
Location: /my/env/lib/pythonx.x/site-packages
Files:
  ../somepackage/__init__.py
  [...]
```

List what packages are outdated:

```
$ pip list --outdated
SomePackage (Current: 1.0 Latest: 2.0)
```

Upgrade a package:

```
$ pip install --upgrade SomePackage
[...]
Found existing installation: SomePackage 1.0
```

```
Uninstalling SomePackage:
  Successfully uninstalled SomePackage
Running setup.py install for SomePackage
Successfully installed SomePackage
```

Uninstall a package:

```
$ pip uninstall SomePackage
Uninstalling SomePackage:
  /my/env/lib/pythonx.x/site-packages/somepackage
Proceed (y/n)? y
Successfully uninstalled SomePackage
```

CHAPTER 2

Installation

2.1 Do I need to install pip?

pip is already installed if you are using Python 2 >=2.7.9 or Python 3 >=3.4 downloaded from python.org or if you are working in a Virtual Environment created by virtualenv or pyvenv. Just make sure to *upgrade pip*.

2.2 Installing with get-pip.py

To install pip, securely download get-pip.py.[1]:

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

Inspect `get-pip.py` for any malevolence. Then run the following:

```
python get-pip.py
```

Warning: Be cautious if you are using a Python install that is managed by your operating system or another package manager. `get-pip.py` does not coordinate with those tools, and may leave your system in an inconsistent state.

`get-pip.py` also installs setuptools[2] and wheel if they are not already. setuptools is required to install source distributions. Both are required in order to build a *Wheel Cache* (which improves installation speed), although neither are required to install pre-built wheels.

Note: The get-pip.py script is supported on the same python version as pip. For the now unsupported Python 2.6, alternate script is available here.

[1] "Secure" in this context means using a modern browser or a tool like *curl* that verifies SSL certificates when downloading from https URLs.
[2] Beginning with pip v1.5.1, `get-pip.py` stopped requiring setuptools to be installed first.

2.2.1 get-pip.py options

--no-setuptools
If set, do not attempt to install setuptools

--no-wheel
If set, do not attempt to install wheel

`get-pip.py` allows *pip install options* and the *general options*. Below are some examples:

Install from local copies of pip and setuptools:

```
python get-pip.py --no-index --find-links=/local/copies
```

Install to the user site[3]:

```
python get-pip.py --user
```

Install behind a proxy:

```
python get-pip.py --proxy="http://[user:passwd@]proxy.server:port"
```

2.3 Using Linux Package Managers

See Installing pip/setuptools/wheel with Linux Package Managers in the Python Packaging User Guide.

2.4 Upgrading pip

On Linux or macOS:

```
pip install -U pip
```

On Windows[4]:

```
python -m pip install -U pip
```

2.5 Python and OS Compatibility

pip works with CPython versions 2.7, 3.3, 3.4, 3.5, 3.6 and also pypy.

This means pip works on the latest patch version of each of these minor versions. Previous patch versions are supported on a best effort approach.

pip works on Unix/Linux, macOS, and Windows.

---

[3] The pip developers are considering making `--user` the default for all installs, including `get-pip.py` installs of pip, but at this time, `--user` installs for pip itself, should not be considered to be fully tested or endorsed. For discussion, see Issue 1668.

[4] https://github.com/pypa/pip/issues/1299

CHAPTER 3

User Guide

Contents

3.1 Running pip

pip is a command line program. When you install pip, a `pip` command is added to your system, which can be run from the command prompt as follows:

```
$ pip <pip arguments>
```

If you cannot run the `pip` command directly (possibly because the location where it was installed isn't on your operating system's `PATH`) then you can run pip via the Python interpreter:

```
$ python -m pip <pip arguments>
```

On Windows, the `py` launcher can be used:

```
$ py -m pip <pip arguments>
```

Even though pip is available from your Python installation as an importable module, via `import pip`, it is *not supported* to use pip in this way. For more details, see *Using pip from your program*.

3.2 Installing Packages

pip supports installing from PyPI, version control, local projects, and directly from distribution files.

The most common scenario is to install from PyPI using *Requirement Specifiers*

```
$ pip install SomePackage            # latest version
$ pip install SomePackage==1.0.4     # specific version
$ pip install 'SomePackage>=1.0.4'     # minimum version
```

For more information and examples, see the *pip install* reference.

3.3 Requirements Files

"Requirements files" are files containing a list of items to be installed using *pip install* like so:

```
pip install -r requirements.txt
```

Details on the format of the files are here: *Requirements File Format*.

Logically, a Requirements file is just a list of *pip install* arguments placed in a file. Note that you should not rely on the items in the file being installed by pip in any particular order.

In practice, there are 4 common uses of Requirements files:

1. Requirements files are used to hold the result from *pip freeze* for the purpose of achieving *repeatable installations*. In this case, your requirement file contains a pinned version of everything that was installed when *pip freeze* was run.

```
pip freeze > requirements.txt
pip install -r requirements.txt
```

2. Requirements files are used to force pip to properly resolve dependencies. As it is now, pip doesn't have true dependency resolution, but instead simply uses the first specification it finds for a project. E.g. if *pkg1* requires *pkg3>=1.0* and *pkg2* requires *pkg3>=1.0,<=2.0*, and if *pkg1* is resolved first, pip will only use *pkg3>=1.0*, and could easily end up installing a version of *pkg3* that conflicts with the needs of *pkg2*. To solve this problem, you can place *pkg3>=1.0,<=2.0* (i.e. the correct specification) into your requirements file directly along with the other top level requirements. Like so:

```
pkg1
pkg2
pkg3>=1.0,<=2.0
```

3. Requirements files are used to force pip to install an alternate version of a sub-dependency. For example, suppose *ProjectA* in your requirements file requires *ProjectB*, but the latest version (v1.3) has a bug, you can force pip to accept earlier versions like so:

```
ProjectA
ProjectB<1.3
```

4. Requirements files are used to override a dependency with a local patch that lives in version control. For example, suppose a dependency, *SomeDependency* from PyPI has a bug, and you can't wait for an upstream fix. You could clone/copy the src, make the fix, and place it in VCS with the tag *sometag*. You'd reference it in your requirements file with a line like so:

```
git+https://myvcs.com/some_dependency@sometag#egg=SomeDependency
```

   If *SomeDependency* was previously a top-level requirement in your requirements file, then **replace** that line with the new line. If *SomeDependency* is a sub-dependency, then **add** the new line.

It's important to be clear that pip determines package dependencies using install_requires metadata, not by discovering *requirements.txt* files embedded in projects.

See also:

- *Requirements File Format*
- *pip freeze*
- "setup.py vs requirements.txt" (an article by Donald Stufft)

3.4 Constraints Files

Constraints files are requirements files that only control which version of a requirement is installed, not whether it is installed or not. Their syntax and contents is nearly identical to *Requirements Files*. There is one key difference: Including a package in a constraints file does not trigger installation of the package.

Use a constraints file like so:

```
pip install -c constraints.txt
```

Constraints files are used for exactly the same reason as requirements files when you don't know exactly what things you want to install. For instance, say that the "helloworld" package doesn't work in your environment, so you have a local patched version. Some things you install depend on "helloworld", and some don't.

One way to ensure that the patched version is used consistently is to manually audit the dependencies of everything you install, and if "helloworld" is present, write a requirements file to use when installing that thing.

Constraints files offer a better way: write a single constraints file for your organisation and use that everywhere. If the thing being installed requires "helloworld" to be installed, your fixed version specified in your constraints file will be used.

Constraints file support was added in pip 7.1.

3.5 Installing from Wheels

"Wheel" is a built, archive format that can greatly speed installation compared to building and installing from source archives. For more information, see the Wheel docs , PEP427, and PEP425

Pip prefers Wheels where they are available. To disable this, use the *–no-binary* flag for *pip install*.

If no satisfactory wheels are found, pip will default to finding source archives.

To install directly from a wheel archive:

```
pip install SomePackage-1.0-py2.py3-none-any.whl
```

For the cases where wheels are not available, pip offers *pip wheel* as a convenience, to build wheels for all your requirements and dependencies.

pip wheel requires the wheel package to be installed, which provides the "bdist_wheel" setuptools extension that it uses.

To build wheels for your requirements and all their dependencies to a local directory:

```
pip install wheel
pip wheel --wheel-dir=/local/wheels -r requirements.txt
```

And *then* to install those requirements just using your local directory of wheels (and not from PyPI):

```
pip install --no-index --find-links=/local/wheels -r requirements.txt
```

3.6 Uninstalling Packages

pip is able to uninstall most packages like so:

```
$ pip uninstall SomePackage
```

pip also performs an automatic uninstall of an old version of a package before upgrading to a newer version.

For more information and examples, see the *pip uninstall* reference.

3.7 Listing Packages

To list installed packages:

```
$ pip list
docutils (0.9.1)
Jinja2 (2.6)
Pygments (1.5)
Sphinx (1.1.2)
```

To list outdated packages, and show the latest version available:

```
$ pip list --outdated
docutils (Current: 0.9.1 Latest: 0.10)
Sphinx (Current: 1.1.2 Latest: 1.1.3)
```

To show details about an installed package:

```
$ pip show sphinx
---
Name: Sphinx
Version: 1.1.3
Location: /my/env/lib/pythonx.x/site-packages
Requires: Pygments, Jinja2, docutils
```

For more information and examples, see the *pip list* and *pip show* reference pages.

3.8 Searching for Packages

pip can search PyPI for packages using the `pip search` command:

```
$ pip search "query"
```

The query will be used to search the names and summaries of all packages.

For more information and examples, see the *pip search* reference.

3.9 Configuration

3.9.1 Config file

pip allows you to set all command line option defaults in a standard ini style config file.

The names and locations of the configuration files vary slightly across platforms. You may have per-user, per-virtualenv or site-wide (shared amongst all users) configuration:

Per-user:

- On Unix the default configuration file is: `$HOME/.config/pip/pip.conf` which respects the `XDG_CONFIG_HOME` environment variable.
- On macOS the configuration file is `$HOME/Library/Application Support/pip/pip.conf` if directory `$HOME/Library/Application Support/pip` exists else `$HOME/.config/pip/pip.conf`.
- On Windows the configuration file is `%APPDATA%\pip\pip.ini`.

There are also a legacy per-user configuration file which is also respected, these are located at:

- On Unix and macOS the configuration file is: `$HOME/.pip/pip.conf`

- On Windows the configuration file is: `%HOME%\pip\pip.ini`

You can set a custom path location for this config file using the environment variable `PIP_CONFIG_FILE`.

Inside a virtualenv:

- On Unix and macOS the file is `$VIRTUAL_ENV/pip.conf`
- On Windows the file is: `%VIRTUAL_ENV%\pip.ini`

Site-wide:

- On Unix the file may be located in `/etc/pip.conf`. Alternatively it may be in a "pip" subdirectory of any of the paths set in the environment variable `XDG_CONFIG_DIRS` (if it exists), for example `/etc/xdg/pip/pip.conf`.
- On macOS the file is: `/Library/Application Support/pip/pip.conf`
- On Windows XP the file is: `C:\Documents and Settings\All Users\Application Data\pip\pip.ini`
- On Windows 7 and later the file is hidden, but writeable at `C:\ProgramData\pip\pip.ini`
- Site-wide configuration is not supported on Windows Vista

If multiple configuration files are found by pip then they are combined in the following order:

1. Firstly the site-wide file is read, then
2. The per-user file is read, and finally
3. The virtualenv-specific file is read.

Each file read overrides any values read from previous files, so if the global timeout is specified in both the site-wide file and the per-user file then the latter value is the one that will be used.

The names of the settings are derived from the long command line option, e.g. if you want to use a different package index (`--index-url`) and set the HTTP timeout (`--default-timeout`) to 60 seconds your config file would look like this:

```
[global]
timeout = 60
index-url = http://download.zope.org/ppix
```

Each subcommand can be configured optionally in its own section so that every global setting with the same name will be overridden; e.g. decreasing the `timeout` to `10` seconds when running the *freeze* (Freezing Requirements) command and using `60` seconds for all other commands is possible with:

```
[global]
timeout = 60

[freeze]
timeout = 10
```

Boolean options like `--ignore-installed` or `--no-dependencies` can be set like this:

```
[install]
ignore-installed = true
no-dependencies = yes
```

To enable the boolean options `--no-compile` and `--no-cache-dir`, falsy values have to be used:

```
[global]
no-cache-dir = false

[install]
no-compile = no
```

Appending options like `--find-links` can be written on multiple lines:

```
[global]
find-links =
    http://download.example.com

[install]
find-links =
    http://mirror1.example.com
    http://mirror2.example.com
```

3.9.2 Environment Variables

pip's command line options can be set with environment variables using the format `PIP_<UPPER_LONG_NAME>` . Dashes (-) have to be replaced with underscores (_).

For example, to set the default timeout:

```
export PIP_DEFAULT_TIMEOUT=60
```

This is the same as passing the option to pip directly:

```
pip --default-timeout=60 [...]
```

For command line options which can be repeated, use a space to separate multiple values. For example:

```
export PIP_FIND_LINKS="http://mirror1.example.com http://mirror2.example.com"
```

is the same as calling:

```
pip install --find-links=http://mirror1.example.com --find-links=http://mirror2.
↪example.com
```

3.9.3 Config Precedence

Command line options have precedence over environment variables, which have precedence over the config file.

Within the config file, command specific sections have precedence over the global section.

Examples:

- `--host=foo` overrides `PIP_HOST=foo`
- `PIP_HOST=foo` overrides a config file with `[global] host = foo`
- A command specific section in the config file `[<command>] host = bar` overrides the option with same name in the `[global]` config file section

3.10 Command Completion

pip comes with support for command line completion in bash, zsh and fish.

To setup for bash:

```
$ pip completion --bash >> ~/.profile
```

To setup for zsh:

```
$ pip completion --zsh >> ~/.zprofile
```

To setup for fish:

```
$ pip completion --fish > ~/.config/fish/completions/pip.fish
```

Alternatively, you can use the result of the `completion` command directly with the eval function of your shell, e.g. by adding the following to your startup file:

```
eval "`pip completion --bash`"
```

3.11 Installing from local packages

In some cases, you may want to install from local packages only, with no traffic to PyPI.

First, download the archives that fulfill your requirements:

```
$ pip install --download DIR -r requirements.txt
```

Note that `pip install --download` will look in your wheel cache first, before trying to download from PyPI. If you've never installed your requirements before, you won't have a wheel cache for those items. In that case, if some of your requirements don't come as wheels from PyPI, and you want wheels, then run this instead:

```
$ pip wheel --wheel-dir DIR -r requirements.txt
```

Then, to install from local only, you'll be using *–find-links* and *–no-index* like so:

```
$ pip install --no-index --find-links=DIR -r requirements.txt
```

3.12 "Only if needed" Recursive Upgrade

`pip install --upgrade` now has a `--upgrade-strategy` option which controls how pip handles upgrading of dependencies. There are 2 upgrade strategies supported:

- `eager`: upgrades all dependencies regardless of whether they still satisfy the new parent requirements
- `only-if-needed`: upgrades a dependency only if it does not satisfy the new parent requirements

The default strategy is `only-if-needed`. This was changed in pip 10.0 due to the breaking nature of `eager` when upgrading conflicting dependencies.

As an historic note, an earlier "fix" for getting the `only-if-needed` behaviour was:

```
pip install --upgrade --no-deps SomePackage
pip install SomePackage
```

A proposal for an `upgrade-all` command is being considered as a safer alternative to the behaviour of eager upgrading.

3.13 User Installs

With Python 2.6 came the "user scheme" for installation, which means that all Python distributions support an alternative install location that is specific to a user. The default location for each OS is explained in the python documentation for the site.USER_BASE variable. This mode of installation can be turned on by specifying the *–user* option to `pip install`.

Moreover, the "user scheme" can be customized by setting the `PYTHONUSERBASE` environment variable, which updates the value of `site.USER_BASE`.

To install "SomePackage" into an environment with site.USER_BASE customized to '/myappenv', do the following:

```
export PYTHONUSERBASE=/myappenv
pip install --user SomePackage
```

`pip install --user` follows four rules:

1. When globally installed packages are on the python path, and they *conflict* with the installation requirements, they are ignored, and *not* uninstalled.
2. When globally installed packages are on the python path, and they *satisfy* the installation requirements, pip does nothing, and reports that requirement is satisfied (similar to how global packages can satisfy requirements when installing packages in a `--system-site-packages` virtualenv).
3. pip will not perform a `--user` install in a `--no-site-packages` virtualenv (i.e. the default kind of virtualenv), due to the user site not being on the python path. The installation would be pointless.
4. In a `--system-site-packages` virtualenv, pip will not install a package that conflicts with a package in the virtualenv site-packages. The –user installation would lack sys.path precedence and be pointless.

To make the rules clearer, here are some examples:

From within a `--no-site-packages` virtualenv (i.e. the default kind):

```
$ pip install --user SomePackage
Can not perform a '--user' install. User site-packages are not visible in this␣
↪virtualenv.
```

From within a `--system-site-packages` virtualenv where `SomePackage==0.3` is already installed in the virtualenv:

```
$ pip install --user SomePackage==0.4
Will not install to the user site because it will lack sys.path precedence
```

From within a real python, where `SomePackage` is *not* installed globally:

```
$ pip install --user SomePackage
[...]
Successfully installed SomePackage
```

From within a real python, where `SomePackage` *is* installed globally, but is *not* the latest version:

```
$ pip install --user SomePackage
[...]
Requirement already satisfied (use --upgrade to upgrade)

$ pip install --user --upgrade SomePackage
[...]
Successfully installed SomePackage
```

From within a real python, where `SomePackage` *is* installed globally, and is the latest version:

```
$ pip install --user SomePackage
[...]
Requirement already satisfied (use --upgrade to upgrade)

$ pip install --user --upgrade SomePackage
[...]
Requirement already up-to-date: SomePackage

# force the install
$ pip install --user --ignore-installed SomePackage
[...]
Successfully installed SomePackage
```

3.14 Ensuring Repeatability

pip can achieve various levels of repeatability:

3.14.1 Pinned Version Numbers

Pinning the versions of your dependencies in the requirements file protects you from bugs or incompatibilities in newly released versions:

```
SomePackage == 1.2.3
DependencyOfSomePackage == 4.5.6
```

Using *pip freeze* to generate the requirements file will ensure that not only the top-level dependencies are included but their sub-dependencies as well, and so on. Perform the installation using *–no-deps* for an extra dose of insurance against installing anything not explicitly listed.

This strategy is easy to implement and works across OSes and architectures. However, it trusts PyPI and the certificate authority chain. It also relies on indices and find-links locations not allowing packages to change without a version increase. (PyPI does protect against this.)

3.14.2 Hash-checking Mode

Beyond pinning version numbers, you can add hashes against which to verify downloaded packages:

```
FooProject == 1.2 --
 ↪hash=sha256:2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

This protects against a compromise of PyPI or the HTTPS certificate chain. It also guards against a package changing without its version number changing (on indexes that allow this). This approach is a good fit for automated server deployments.

Hash-checking mode is a labor-saving alternative to running a private index server containing approved packages: it removes the need to upload packages, maintain ACLs, and keep an audit trail (which a VCS gives you on the requirements file for free). It can also substitute for a vendor library, providing easier upgrades and less VCS noise. It does not, of course, provide the availability benefits of a private index or a vendor library.

For more, see *pip install's discussion of hash-checking mode*.

3.14.3 Installation Bundles

Using *pip wheel*, you can bundle up all of a project's dependencies, with any compilation done, into a single archive. This allows installation when index servers are unavailable and avoids time-consuming recompilation. Create an archive like this:

```
$ tempdir=$(mktemp -d /tmp/wheelhouse-XXXXX)
$ pip wheel -r requirements.txt --wheel-dir=$tempdir
$ cwd=`pwd`
$ (cd "$tempdir"; tar -cjvf "$cwd/bundled.tar.bz2" *)
```

You can then install from the archive like this:

```
$ tempdir=$(mktemp -d /tmp/wheelhouse-XXXXX)
$ (cd $tempdir; tar -xvf /path/to/bundled.tar.bz2)
$ pip install --force-reinstall --ignore-installed --upgrade --no-index --no-deps
↪$tempdir/*
```

Note that compiled packages are typically OS- and architecture-specific, so these archives are not necessarily portable across machines.

Hash-checking mode can be used along with this method to ensure that future archives are built with identical packages.

Warning: Finally, beware of the `setup_requires` keyword arg in `setup.py`. The (rare) packages that use it will cause those dependencies to be downloaded by setuptools directly, skipping pip's protections. If you need to use such a package, see *Controlling setup_requires*.

3.15 Using pip from your program

As noted previously, pip is a command line program. While it is implemented in Python, and so is available from your Python code via `import pip`, you must not use pip's internal APIs in this way. There are a number of reasons for this:

1. The pip code assumes that is in sole control of the global state of the program. Pip manages things like the logging system configuration, or the values of the standard IO streams, without considering the possibility that user code might be affected.
2. Pip's code is *not* thread safe. If you were to run pip in a thread, there is no guarantee that either your code or pip's would work as you expect.
3. Pip assumes that once it has finished its work, the process will terminate. It doesn't need to handle the possibility that other code will continue to run after that point, so (for example) calling pip twice in the same process is likely to have issues.

This does not mean that the pip developers are opposed in principle to the idea that pip could be used as a library - it's just that this isn't how it was written, and it would be a lot of work to redesign the internals for use as a library,

handling all of the above issues, and designing a usable, robust and stable API that we could guarantee would remain available across multiple releases of pip. And we simply don't currently have the resources to even consider such a task.

What this means in practice is that everything inside of pip is considered an implementation detail. Even the fact that the import name is `pip` is subject to change without notice. While we do try not to break things as much as possible, all the internal APIs can change at any time, for any reason. It also means that we generally *won't* fix issues that are a result of using pip in an unsupported way.

It should also be noted that installing packages into `sys.path` in a running Python process is something that should only be done with care. The import system caches certain data, and installing new packages while a program is running may not always behave as expected. In practice, there is rarely an issue, but it is something to be aware of.

Having said all of the above, it is worth covering the options available if you decide that you do want to run pip from within your program. The most reliable approach, and the one that is fully supported, is to run pip in a subprocess. This is easily done using the standard `subprocess` module:

```
subprocess.check_call([sys.executable, '-m', 'pip', 'install', 'my_package'])
```

If you want to process the output further, use one of the other APIs in the module:

```
reqs = subprocess.check_output([sys.executable, '-m', 'pip', 'freeze'])
```

If you don't want to use pip's command line functionality, but are rather trying to implement code that works with Python packages, their metadata, or PyPI, then you should consider other, supported, packages that offer this type of ability. Some examples that you could consider include:

- `packaging` - Utilities to work with standard package metadata (versions, requirements, etc.)
- `setuptools` (specifically `pkg_resources`) - Functions for querying what packages the user has installed on their system.
- `distlib` - Packaging and distribution utilities (including functions for interacting with PyPI).

CHAPTER 4

Reference Guide

4.1 pip

Contents

4.1.1 Usage

```
pip <command> [options]
```

4.1.2 Description

Logging

Console logging

pip offers *-v, –verbose* and *-q, –quiet* to control the console log level. By default, some messages (error and warnings) are colored in the terminal. If you want to suppress the colored output use *–no-color*.

File logging

pip offers the *–log* option for specifying a file where a maximum verbosity log will be kept. This option is empty by default. This log appends to previous logging.

Like all pip options, `--log` can also be set as an environment variable, or placed into the pip config file. See the *Configuration* section.

–exists-action option

This option specifies default behavior when path already exists. Possible cases: downloading files or checking out repositories for installation, creating archives. If `--exists-action` is not defined, pip will prompt when decision is needed.

(s)witch Only relevant to VCS checkout. Attempt to switch the checkout to the appropriate url and/or revision.

(i)gnore Abort current operation (e.g. don't copy file, don't create archive, don't modify a checkout).

(w)ipe Delete the file or VCS checkout before trying to create, download, or checkout a new one.

(b)ackup Rename the file or checkout to `{name}{'.bak' * n}`, where n is some number of `.bak` extensions, such that the file didn't exist at some point. So the most recent backup will be the one with the largest number after `.bak`.

(a)bort Abort pip and return non-zero exit status.

Build System Interface

Pip builds packages by invoking the build system. Presently, the only supported build system is `setuptools`, but in the future, pip will support PEP517 which allows projects to specify an alternative build system in a `pyproject.toml` file. As well as package building, the build system is also invoked to install packages direct from source. This is handled by invoking the build system to build a wheel, and then installing from that wheel. The built wheel is cached locally by pip to avoid repeated identical builds.

The current interface to the build system is via the `setup.py` command line script - all build actions are defined in terms of the specific `setup.py` command line that will be run to invoke the required action.

Setuptools Injection

As noted above, the supported build system is `setuptools`. However, not all packages use `setuptools` in their build scripts. To support projects that use "pure `distutils`", pip injects `setuptools` into `sys.modules` before invoking `setup.py`. The injection should be transparent to `distutils`-based projects, but 3rd party build tools wishing to provide a `setup.py` emulating the commands pip requires may need to be aware that it takes place.

Build System Output

Any output produced by the build system will be read by pip (for display to the user if requested). In order to correctly read the build system output, pip requires that the output is written in a well-defined encoding, specifically the encoding the user has configured for text output (which can be obtained in Python using `locale.getpreferredencoding`). If the configured encoding is ASCII, pip assumes UTF-8 (to account for the behaviour of some Unix systems).

Build systems should ensure that any tools they invoke (compilers, etc) produce output in the correct encoding. In practice - and in particular on Windows, where tools are inconsistent in their use of the "OEM" and "ANSI" codepages - this may not always be possible. Pip will therefore attempt to recover cleanly if presented with incorrectly encoded build tool output, by translating unexpected byte sequences to Python-style hexadecimal escape sequences (`"\x80\xff"`, etc). However, it is still possible for output to be displayed using an incorrect encoding (mojibake).

PEP 518 Support

Pip supports projects declaring dependencies that are required at install time using a `pyproject.toml` file, in the form described in PEP518. When building a project, pip will install the required dependencies locally, and make them available to the build process.

As noted in the PEP, the minimum requirements for pip to be able to build a project are:

```
[build-system]
# Minimum requirements for the build system to execute.
requires = ["setuptools", "wheel"]
```

`setuptools` and `wheel` **must** be included in any `pyproject.toml` provided by a project - pip will assume these as a default, but will not add them to an explicitly supplied list in a project supplied `pyproject.toml` file. Once PEP517 support is added, this restriction will be lifted and alternative build tools will be allowed.

When making build requirements available, pip does so in an *isolated environment*. That is, pip does not install those requirements into the user's `site-packages`, but rather installs them in a temporary directory which it adds to the user's `sys.path` for the duration of the build. This ensures that build requirements are handled independently of the user's runtime environment. For example, a project that needs a recent version of setuptools to build can still be installed, even if the user has an older version installed (and without silently replacing that version).

In certain cases, projects (or redistributors) may have workflows that explicitly manage the build environment. For such workflows, build isolation can be problematic. If this is the case, pip provides a `--no-build-isolation` flag to disable build isolation. Users supplying this flag are responsible for ensuring the build environment is managed appropriately.

The current implementation of PEP518 in pip requires that any dependencies specified in `pyproject.toml` are available as wheels. This is a technical limitation of the implementation - dependencies only available as source would require a build step of their own, which would recursively invoke the PEP518 dependency installation process. The potentially unbounded recursion involved was not considered acceptable, and so installation of build dependencies from source has been disabled until a safe resolution of this issue has been found.

Future Developments

PEP426 notes that the intention is to add hooks to project metadata in version 2.1 of the metadata spec, to explicitly define how to build a project from its source. Once this version of the metadata spec is final, pip will migrate to using that interface. At that point, the `setup.py` interface documented here will be retained solely for legacy purposes, until projects have migrated.

Specifically, applications should *not* expect to rely on there being any form of backward compatibility guarantees around the `setup.py` interface.

Build Options

The `--global-option` and `--build-option` arguments to the `pip install` and `pip wheel` inject additional arguments into the `setup.py` command (`--build-option` is only available in `pip wheel`). These arguments are included in the command as follows:

```
python setup.py <global_options> BUILD COMMAND <build_options>
```

The options are passed unmodified, and presently offer direct access to the distutils command line. Use of `--global-option` and `--build-option` should be considered as build system dependent, and may not be supported in the current form if support for alternative build systems is added to pip.

4.1.3 General Options

-h, --help
Show help.

--isolated
Run pip in an isolated mode, ignoring environment variables and user configuration.

-v, --verbose
Give more output. Option is additive, and can be used up to 3 times.

-V, --version
Show version and exit.

-q, --quiet
Give less output. Option is additive, and can be used up to 3 times (corresponding to WARNING, ERROR, and CRITICAL logging levels).

--log <path>
Path to a verbose appending log.

--proxy <proxy>
Specify a proxy in the form [user:passwd@]proxy.server:port.

--retries <retries>
Maximum number of retries each connection should attempt (default 5 times).

--timeout <sec>
Set the socket timeout (default 15 seconds).

--exists-action <action>
Default action when a path already exists: (s)witch, (i)gnore, (w)ipe, (b)ackup, (a)bort).

--trusted-host <hostname>
Mark this host as trusted, even though it does not have valid or any HTTPS.

--cert <path>
Path to alternate CA bundle.

--client-cert <path>
Path to SSL client certificate, a single file containing the private key and the certificate in PEM format.

--cache-dir <dir>
Store the cache data in <dir>.

--no-cache-dir
Disable the cache.

--disable-pip-version-check
Don't periodically check PyPI to determine whether a new version of pip is available for download. Implied with –no-index.

--no-color
Suppress colored output

4.2 pip install

Contents

4.2.1 Usage

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

4.2.2 Description

Install packages from:

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

pip also supports installing from "requirements files", which provide an easy way to specify a whole environment to be installed.

Overview

Pip install has several stages:

1. Identify the base requirements. The user supplied arguments are processed here.
2. Resolve dependencies. What will be installed is determined here.
3. Build wheels. All the dependencies that can be are built into wheels.
4. Install the packages (and uninstall anything being upgraded/replaced).

Argument Handling

When looking at the items to be installed, pip checks what type of item each is, in the following order:

1. Project or archive URL.
2. Local directory (which must contain a `setup.py`, or pip will report an error).
3. Local file (a sdist or wheel format archive, following the naming conventions for those formats).
4. A requirement, as specified in PEP 440.

Each item identified is added to the set of requirements to be satisfied by the install.

Working Out the Name and Version

For each candidate item, pip needs to know the project name and version. For wheels (identified by the `.whl` file extension) this can be obtained from the filename, as per the Wheel spec. For local directories, or explicitly specified sdist files, the `setup.py egg_info` command is used to determine the project metadata. For sdists located via an index, the filename is parsed for the name and project version (this is in theory slightly less reliable than using the `egg_info` command, but avoids downloading and processing unnecessary numbers of files).

Any URL may use the `#egg=name` syntax (see *VCS Support*) to explicitly state the project name.

Satisfying Requirements

Once pip has the set of requirements to satisfy, it chooses which version of each requirement to install using the simple rule that the latest version that satisfies the given constraints will be installed (but see *here* for an exception regarding pre-release versions). Where more than one source of the chosen version is available, it is assumed that any source is acceptable (as otherwise the versions would differ).

Installation Order

As of v6.1.0, pip installs dependencies before their dependents, i.e. in "topological order". This is the only commitment pip currently makes related to order. While it may be coincidentally true that pip will install things in the order of the install arguments or in the order of the items in a requirements file, this is not a promise.

In the event of a dependency cycle (aka "circular dependency"), the current implementation (which might possibly change later) has it such that the first encountered member of the cycle is installed last.

For instance, if quux depends on foo which depends on bar which depends on baz, which depends on foo:

```
pip install quux
...
Installing collected packages baz, bar, foo, quux

pip install bar
...
Installing collected packages foo, baz, bar
```

Prior to v6.1.0, pip made no commitments about install order.

The decision to install topologically is based on the principle that installations should proceed in a way that leaves the environment usable at each step. This has two main practical benefits:

1. Concurrent use of the environment during the install is more likely to work.

2. A failed install is less likely to leave a broken environment. Although pip would like to support failure rollbacks eventually, in the mean time, this is an improvement.

Although the new install order is not intended to replace (and does not replace) the use of `setup_requires` to declare build dependencies, it may help certain projects install from sdist (that might previously fail) that fit the following profile:

1. They have build dependencies that are also declared as install dependencies using `install_requires`.
2. `python setup.py egg_info` works without their build dependencies being installed.
3. For whatever reason, they don't or won't declare their build dependencies using `setup_requires`.

Requirements File Format

Each line of the requirements file indicates something to be installed, and like arguments to *pip install*, the following forms are supported:

```
[[--option]...]
<requirement specifier> [; markers] [[--option]...]
<archive url/path>
[-e] <local project path>
[-e] <vcs project url>
```

For details on requirement specifiers, see *Requirement Specifiers*.

See the *pip install Examples* for examples of all these forms.

A line that begins with # is treated as a comment and ignored. Whitespace followed by a # causes the # and the remainder of the line to be treated as a comment.

A line ending in an unescaped \ is treated as a line continuation and the newline following it is effectively ignored.

Comments are stripped *before* line continuations are processed.

The following options are supported:

- *-i, –index-url*
- *–extra-index-url*
- *–no-index*
- *-f, –find-links*
- *–no-binary*
- *–only-binary*
- *–require-hashes*

For example, to specify *–no-index* and 2 *–find-links* locations:

```
--no-index
--find-links /my/local/archives
--find-links http://some.archives.com/archives
```

If you wish, you can refer to other requirements files, like this:

```
-r more_requirements.txt
```

You can also refer to *constraints files*, like this:

```
-c some_constraints.txt
```

Using Environment Variables

Since version 10, pip supports the use of environment variables inside the requirements file. You can now store sensitive data (tokens, keys, etc.) in environment variables and only specify the variable name for your requirements, letting pip lookup the value at runtime. This approach aligns with the commonly used 12-factor configuration pattern.

You have to use the POSIX format for variable names including brackets around the uppercase name as shown in this example: `${API_TOKEN}`. pip will attempt to find the corresponding environment variable defined on the host system at runtime.

Note: There is no support for other variable expansion syntaxes such as `$VARIABLE` and `%VARIABLE%`.

Example Requirements File

Use `pip install -r example-requirements.txt` to install:

```
#
####### example-requirements.txt #######
#
###### Requirements without Version Specifiers ######
nose
nose-cov
beautifulsoup4
#
###### Requirements with Version Specifiers ######
#   See https://www.python.org/dev/peps/pep-0440/#version-specifiers
docopt == 0.6.1             # Version Matching. Must be version 0.6.1
keyring >= 4.1.1            # Minimum version 4.1.1
coverage != 3.5             # Version Exclusion. Anything except version 3.5
Mopidy-Dirble ~= 1.1        # Compatible release. Same as >= 1.1, == 1.*
#
###### Refer to other requirements files ######
-r other-requirements.txt
#
#
###### A particular file ######
./downloads/numpy-1.9.2-cp34-none-win32.whl
http://wxpython.org/Phoenix/snapshot-builds/wxPython_Phoenix-3.0.3.dev1820+49a8884-
↪cp34-none-win_amd64.whl
#
###### Additional Requirements without Version Specifiers ######
#   Same as 1st section, just here to show that you can put things in any order.
rejected
green
#
```

Requirement Specifiers

pip supports installing from a package index using a requirement specifier. Generally speaking, a requirement specifier is composed of a project name followed by optional version specifiers. PEP508 contains a full specification of the

format of a requirement (`pip` does not support the `url_req` form of specifier at this time).

Some examples:

```
SomeProject
SomeProject == 1.3
SomeProject >=1.2,<.2.0
SomeProject[foo, bar]
SomeProject~=1.4.2
```

Since version 6.0, pip also supports specifiers containing environment markers like so:

```
SomeProject ==5.4 ; python_version < '2.7'
SomeProject; sys_platform == 'win32'
```

Environment markers are supported in the command line and in requirements files.

Note: Use quotes around specifiers in the shell when using >, <, or when using environment markers. Don't use quotes in requirement files.[1]

Per-requirement Overrides

Since version 7.0 pip supports controlling the command line options given to `setup.py` via requirements files. This disables the use of wheels (cached or otherwise) for that package, as `setup.py` does not exist for wheels.

The `--global-option` and `--install-option` options are used to pass options to `setup.py`. For example:

```
FooProject >= 1.2 --global-option="--no-user-cfg" \
                  --install-option="--prefix='/usr/local'" \
                  --install-option="--no-compile"
```

The above translates roughly into running FooProject's `setup.py` script as:

```
python setup.py --no-user-cfg install --prefix='/usr/local' --no-compile
```

Note that the only way of giving more than one option to `setup.py` is through multiple `--global-option` and `--install-option` options, as shown in the example above. The value of each option is passed as a single argument to the `setup.py` script. Therefore, a line such as the following is invalid and would result in an installation error.

```
# Invalid. Please use '--install-option' twice as shown above.
FooProject >= 1.2 --install-option="--prefix=/usr/local --no-compile"
```

Pre-release Versions

Starting with v1.4, pip will only install stable versions as specified by PEP426 by default. If a version cannot be parsed as a compliant PEP426 version then it is assumed to be a pre-release.

If a Requirement specifier includes a pre-release or development version (e.g. `>=0.0.dev0`) then pip will allow pre-release and development versions for that requirement. This does not include the != flag.

The `pip install` command also supports a *–pre* flag that will enable installing pre-releases and development releases.

[1] This is true with the exception that pip v7.0 and v7.0.1 required quotes around specifiers containing environment markers in requirement files.

VCS Support

pip supports installing from Git, Mercurial, Subversion and Bazaar, and detects the type of VCS using url prefixes: "git+", "hg+", "bzr+", "svn+".

pip requires a working VCS command on your path: git, hg, svn, or bzr.

VCS projects can be installed in *editable mode* (using the *–editable* option) or not.

- For editable installs, the clone location by default is "<venv path>/src/SomeProject" in virtual environments, and "<cwd>/src/SomeProject" for global installs. The *–src* option can be used to modify this location.
- For non-editable installs, the project is built locally in a temp dir and then installed normally. Note that if a satisfactory version of the package is already installed, the VCS source will not overwrite it without an *–upgrade* flag. VCS requirements pin the package version (specified in the *setup.py* file) of the target commit, not necessarily the commit itself.
- The *pip freeze* subcommand will record the VCS requirement specifier (referencing a specific commit) if and only if the install is done using the editable option.

The "project name" component of the url suffix "egg=<project name>-<version>" is used by pip in its dependency logic to identify the project prior to pip downloading and analyzing the metadata. The optional "version" component of the egg name is not functionally important. It merely provides a human-readable clue as to what version is in use. For projects where setup.py is not in the root of project, "subdirectory" component is used. Value of "subdirectory" component should be a path starting from root of the project to where setup.py is located.

So if your repository layout is:

- pkg_dir/
  - setup.py # setup.py for package `pkg`
  - some_module.py
- other_dir/
  - some_file
- some_other_file

You'll need to use `pip install -e vcs+protocol://repo_url/#egg=pkg&subdirectory=pkg_dir`.

Git

pip currently supports cloning over `git`, `git+http`, `git+https`, `git+ssh`, `git+git` and `git+file`:

Here are the supported forms:

```
[-e] git://git.example.com/MyProject#egg=MyProject
[-e] git+http://git.example.com/MyProject#egg=MyProject
[-e] git+https://git.example.com/MyProject#egg=MyProject
[-e] git+ssh://git.example.com/MyProject#egg=MyProject
[-e] git+git://git.example.com/MyProject#egg=MyProject
[-e] git+file:///home/user/projects/MyProject#egg=MyProject
-e git+git@git.example.com:MyProject#egg=MyProject
```

Passing branch names, a commit hash or a tag name is possible like so:

```
[-e] git://git.example.com/MyProject.git@master#egg=MyProject
[-e] git://git.example.com/MyProject.git@v1.0#egg=MyProject
[-e] git://git.example.com/MyProject.git@da39a3ee5e6b4b0d3255bfef95601890afd80709
↪#egg=MyProject
```

When passing a commit hash, specifying a full hash is preferable to a partial hash because a full hash allows pip to operate more efficiently (e.g. by making fewer network calls).

Mercurial

The supported schemes are: `hg+http`, `hg+https`, `hg+static-http` and `hg+ssh`.

Here are the supported forms:

```
[-e] hg+http://hg.myproject.org/MyProject#egg=MyProject
[-e] hg+https://hg.myproject.org/MyProject#egg=MyProject
[-e] hg+ssh://hg.myproject.org/MyProject#egg=MyProject
[-e] hg+file:///home/user/projects/MyProject#egg=MyProject
```

You can also specify a revision number, a revision hash, a tag name or a local branch name like so:

```
[-e] hg+http://hg.example.com/MyProject@da39a3ee5e6b#egg=MyProject
[-e] hg+http://hg.example.com/MyProject@2019#egg=MyProject
[-e] hg+http://hg.example.com/MyProject@v1.0#egg=MyProject
[-e] hg+http://hg.example.com/MyProject@special_feature#egg=MyProject
```

Subversion

pip supports the URL schemes `svn`, `svn+svn`, `svn+http`, `svn+https`, `svn+ssh`.

You can also give specific revisions to an SVN URL, like so:

```
[-e] svn+svn://svn.example.com/svn/MyProject#egg=MyProject
[-e] svn+http://svn.example.com/svn/MyProject/trunk@2019#egg=MyProject
```

which will check out revision 2019. `@{20080101}` would also check out the revision from 2008-01-01. You can only check out specific revisions using `-e svn+...`.

Bazaar

pip supports Bazaar using the `bzr+http`, `bzr+https`, `bzr+ssh`, `bzr+sftp`, `bzr+ftp` and `bzr+lp` schemes.

Here are the supported forms:

```
[-e] bzr+http://bzr.example.com/MyProject/trunk#egg=MyProject
[-e] bzr+sftp://user@example.com/MyProject/trunk#egg=MyProject
[-e] bzr+ssh://user@example.com/MyProject/trunk#egg=MyProject
[-e] bzr+ftp://user@example.com/MyProject/trunk#egg=MyProject
[-e] bzr+lp:MyProject#egg=MyProject
```

Tags or revisions can be installed like so:

```
[-e] bzr+https://bzr.example.com/MyProject/trunk@2019#egg=MyProject
[-e] bzr+http://bzr.example.com/MyProject/trunk@v1.0#egg=MyProject
```

Using Environment Variables

Since version 10, pip also makes it possible to use environment variables which makes it possible to reference private repositories without having to store access tokens in the requirements file. For example, a private git repository allowing Basic Auth for authentication can be refenced like this:

```
[-e] git+http://${AUTH_USER}:${AUTH_PASSWORD}@git.example.com/MyProject#egg=MyProject
[-e] git+https://${AUTH_USER}:${AUTH_PASSWORD}@git.example.com/MyProject#egg=MyProject
```

Note: Only `${VARIABLE}` is supported, other formats like `$VARIABLE` or `%VARIABLE%` won't work.

Finding Packages

pip searches for packages on PyPI using the http simple interface, which is documented here and there

pip offers a number of Package Index Options for modifying how packages are found.

pip looks for packages in a number of places, on PyPI (if not disabled via `--no-index`), in the local filesystem, and in any additional repositories specified via `--find-links` or `--index-url`. There is no ordering in the locations that are searched, rather they are all checked, and the "best" match for the requirements (in terms of version number - see PEP440 for details) is selected.

See the *pip install Examples*.

SSL Certificate Verification

Starting with v1.3, pip provides SSL certificate verification over https, to prevent man-in-the-middle attacks against PyPI downloads.

Caching

Starting with v6.0, pip provides an on-by-default cache which functions similarly to that of a web browser. While the cache is on by default and is designed do the right thing by default you can disable the cache and always access PyPI by utilizing the `--no-cache-dir` option.

When making any HTTP request pip will first check its local cache to determine if it has a suitable response stored for that request which has not expired. If it does then it simply returns that response and doesn't make the request.

If it has a response stored, but it has expired, then it will attempt to make a conditional request to refresh the cache which will either return an empty response telling pip to simply use the cached item (and refresh the expiration timer) or it will return a whole new response which pip can then store in the cache.

When storing items in the cache, pip will respect the `CacheControl` header if it exists, or it will fall back to the `Expires` header if that exists. This allows pip to function as a browser would, and allows the index server to communicate to pip how long it is reasonable to cache any particular item.

While this cache attempts to minimize network activity, it does not prevent network access altogether. If you want a local install solution that circumvents accessing PyPI, see *Installing from local packages*.

The default location for the cache directory depends on the Operating System:

Unix `~/.cache/pip` and it respects the `XDG_CACHE_HOME` directory.

macOS `~/Library/Caches/pip`.

Windows `<CSIDL_LOCAL_APPDATA>\pip\Cache`

Wheel Cache

Pip will read from the subdirectory `wheels` within the pip cache directory and use any packages found there. This is disabled via the same `--no-cache-dir` option that disables the HTTP cache. The internal structure of that is not part of the pip API. As of 7.0, pip makes a subdirectory for each sdist that wheels are built from and places the resulting wheels inside.

Pip attempts to choose the best wheels from those built in preference to building a new wheel. Note that this means when a package has both optional C extensions and builds *py* tagged wheels when the C extension can't be built that pip will not attempt to build a better wheel for Pythons that would have supported it, once any generic wheel is built. To correct this, make sure that the wheels are built with Python specific tags - e.g. pp on PyPy.

When no wheels are found for an sdist, pip will attempt to build a wheel automatically and insert it into the wheel cache.

Hash-Checking Mode

Since version 8.0, pip can check downloaded package archives against local hashes to protect against remote tampering. To verify a package against one or more hashes, add them to the end of the line:

```
FooProject == 1.2 --
↪hash=sha256:2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824 \
                  --
↪hash=sha256:486ea46224d1bb4fb680f34f7c9ad96a8f24ec88be73ea8e5a6c65260e9cb8a7
```

(The ability to use multiple hashes is important when a package has both binary and source distributions or when it offers binary distributions for a variety of platforms.)

The recommended hash algorithm at the moment is sha256, but stronger ones are allowed, including all those supported by `hashlib`. However, weaker ones such as md5, sha1, and sha224 are excluded to avoid giving a false sense of security.

Hash verification is an all-or-nothing proposition. Specifying a `--hash` against any requirement not only checks that hash but also activates a global *hash-checking mode*, which imposes several other security restrictions:

- Hashes are required for all requirements. This is because a partially-hashed requirements file is of little use and thus likely an error: a malicious actor could slip bad code into the installation via one of the unhashed requirements. Note that hashes embedded in URL-style requirements via the `#md5=...` syntax suffice to satisfy this rule (regardless of hash strength, for legacy reasons), though you should use a stronger hash like sha256 whenever possible.
- Hashes are required for all dependencies. An error results if there is a dependency that is not spelled out and hashed in the requirements file.
- Requirements that take the form of project names (rather than URLs or local filesystem paths) must be pinned to a specific version using ==. This prevents a surprising hash mismatch upon the release of a new version that matches the requirement specifier.
- `--egg` is disallowed, because it delegates installation of dependencies to setuptools, giving up pip's ability to enforce any of the above.

Hash-checking mode can be forced on with the `--require-hashes` command-line option:

```
$ pip install --require-hashes -r requirements.txt
    ...
```

```
    Hashes are required in --require-hashes mode (implicitly on when a hash is
    specified for any package). These requirements were missing hashes,
    leaving them open to tampering. These are the hashes the downloaded
    archives actually had. You can add lines like these to your requirements
    files to prevent tampering.
        pyelasticsearch==1.0 --
↪hash=sha256:44ddfb1225054d7d6b1d02e9338e7d4809be94edbe9929a2ec0807d38df993fa
        more-itertools==2.2 --
↪hash=sha256:93e62e05c7ad3da1a233def6731e8285156701e3419a5fe279017c429ec67ce0
```

This can be useful in deploy scripts, to ensure that the author of the requirements file provided hashes. It is also a convenient way to bootstrap your list of hashes, since it shows the hashes of the packages it fetched. It fetches only the preferred archive for each package, so you may still need to add hashes for alternatives archives using *pip hash*: for instance if there is both a binary and a source distribution.

The *wheel cache* is disabled in hash-checking mode to prevent spurious hash mismatch errors. These would otherwise occur while installing sdists that had already been automatically built into cached wheels: those wheels would be selected for installation, but their hashes would not match the sdist ones from the requirements file. A further complication is that locally built wheels are nondeterministic: contemporary modification times make their way into the archive, making hashes unpredictable across machines and cache flushes. Compilation of C code adds further nondeterminism, as many compilers include random-seeded values in their output. However, wheels fetched from index servers are the same every time. They land in pip's HTTP cache, not its wheel cache, and are used normally in hash-checking mode. The only downside of having the wheel cache disabled is thus extra build time for sdists, and this can be solved by making sure pre-built wheels are available from the index server.

Hash-checking mode also works with *pip download* and *pip wheel*. A *comparison of hash-checking mode with other repeatability strategies* is available in the User Guide.

Warning: Beware of the `setup_requires` keyword arg in `setup.py`. The (rare) packages that use it will cause those dependencies to be downloaded by setuptools directly, skipping pip's hash-checking. If you need to use such a package, see *Controlling setup_requires*.

Warning: Be careful not to nullify all your security work when you install your actual project by using setuptools directly: for example, by calling `python setup.py install`, `python setup.py develop`, or `easy_install`. Setuptools will happily go out and download, unchecked, anything you missed in your requirements file—and it's easy to miss things as your project evolves. To be safe, install your project using pip and *–no-deps*.

Instead of `python setup.py develop`, use...

```
pip install --no-deps -e .
```

Instead of `python setup.py install`, use...

```
pip install --no-deps .
```

Hashes from PyPI

PyPI provides an MD5 hash in the fragment portion of each package download URL, like `#md5=123...`, which pip checks as a protection against download corruption. Other hash algorithms that have guaranteed support from `hashlib` are also supported here: sha1, sha224, sha384, sha256, and sha512. Since this hash originates remotely, it is not a useful guard against tampering and thus does not satisfy the `--require-hashes` demand that every

package have a local hash.

"Editable" Installs

"Editable" installs are fundamentally "setuptools develop mode" installs.

You can install local projects or VCS projects in "editable" mode:

```
$ pip install -e path/to/SomeProject
$ pip install -e git+http://repo/my_project.git#egg=SomeProject
```

(See the *VCS Support* section above for more information on VCS-related syntax.)

For local projects, the "SomeProject.egg-info" directory is created relative to the project path. This is one advantage over just using `setup.py develop`, which creates the "egg-info" directly relative the current working directory.

Controlling setup_requires

Setuptools offers the `setup_requires` setup() keyword for specifying dependencies that need to be present in order for the *setup.py* script to run. Internally, Setuptools uses `easy_install` to fulfill these dependencies.

pip has no way to control how these dependencies are located. None of the Package Index Options have an effect.

The solution is to configure a "system" or "personal" Distutils configuration file to manage the fulfillment.

For example, to have the dependency located at an alternate index, add this:

```
[easy_install]
index_url = https://my.index-mirror.com
```

To have the dependency located from a local directory and not crawl PyPI, add this:

```
[easy_install]
allow_hosts = ''
find_links = file:///path/to/local/archives/
```

Build System Interface

In order for pip to install a package from source, `setup.py` must implement the following commands:

```
setup.py egg_info [--egg-base XXX]
setup.py install --record XXX [--single-version-externally-managed] [--root XXX] [--
↪compile|--no-compile] [--install-headers XXX]
```

The `egg_info` command should create egg metadata for the package, as described in the setuptools documentation at https://setuptools.readthedocs.io/en/latest/setuptools.html#egg-info-create-egg-metadata-and-set-build-tags

The `install` command should implement the complete process of installing the package to the target directory XXX.

To install a package in "editable" mode (`pip install -e`), `setup.py` must implement the following command:

```
setup.py develop --no-deps
```

This should implement the complete process of installing the package in "editable" mode.

All packages will be attempted to built into wheels:

```
setup.py bdist_wheel -d XXX
```

One further `setup.py` command is invoked by `pip install`:

```
setup.py clean
```

This command is invoked to clean up temporary commands from the build. (TODO: Investigate in more detail when this command is required).

No other build system commands are invoked by the `pip install` command.

Installing a package from a wheel does not invoke the build system at all.

4.2.3 Options

-r, --requirement <file>
Install from the given requirements file. This option can be used multiple times.

-c, --constraint <file>
Constrain versions using the given constraints file. This option can be used multiple times.

--no-deps
Don't install package dependencies).

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

-e, --editable <path/url>
Install a project in editable mode (i.e. setuptools "develop mode") from a local project path or a VCS url.

-t, --target <dir>
Install packages into <dir>. By default this will not replace existing files/folders in <dir>. Use –upgrade to replace existing packages in <dir> with new versions.

--user
Install to the Python user install directory for your platform. Typically ~/.local/, or %APPDATA%Python on Windows. (See the Python documentation for site.USER_BASE for full details.)

--root <dir>
Install everything relative to this alternate root directory.

--prefix <dir>
Installation prefix where lib, bin and other top-level folders are placed

-b, --build <dir>
Directory to unpack packages into and build in. Note that an initial build still takes place in a temporary directory. The location of temporary directories can be controlled by setting the TMPDIR environment variable (TEMP on Windows) appropriately.

--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".

-U, --upgrade
Upgrade all specified packages to the newest available version. The handling of dependencies depends on the upgrade-strategy used.

--upgrade-strategy <upgrade_strategy>
Determines how dependency upgrading should be handled (default: %(default)s). "eager" - dependencies are

upgraded regardless of whether the currently installed version satisfies the requirements of the upgraded package(s). "only-if-needed" - are upgraded only when they do not satisfy the requirements of the upgraded package(s).

--force-reinstall
Reinstall all packages even if they are already up-to-date.

-I, --ignore-installed
Ignore the installed packages (reinstalling instead).

--ignore-requires-python
Ignore the Requires-Python information.

--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.

--install-option <options>
Extra arguments to be supplied to the setup.py install command (use like –install-option="–install-scripts=/usr/local/bin"). Use multiple –install-option options to pass multiple options to setup.py install. If you are using an option with a directory path, be sure to use absolute path.

--global-option <options>
Extra global options to be supplied to the setup.py call before the install command.

--compile
Compile Python source files to bytecode

--no-compile
Do not compile Python source files to bytecode

--no-warn-script-location
Do not warn when installing scripts outside PATH

--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, or one or more package names with commas between them. Note that some packages are tricky to compile and may fail to install when this option is used on them.

--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.

--no-clean
Don't clean up build directories).

--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.

--progress-bar <progress_bar>
Specify type of progress to be displayed [on|asciiloff|pretty|emoji] (default: on)

-i, --index-url <url>
Base URL of Python Package Index (default https://pypi.python.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.

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

--no-index
Ignore package index (only looking at –find-links URLs instead).

-f, --find-links <url>
If a url or path to an html file, then parse for links to archives. If a local path or file:// url that's a directory, then look for archives in the directory listing.

--process-dependency-links
Enable the processing of dependency links.

4.2.4 Examples

1. Install *SomePackage* and its dependencies from PyPI using *Requirement Specifiers*

```
$ pip install SomePackage            # latest version
$ pip install SomePackage==1.0.4     # specific version
$ pip install 'SomePackage>=1.0.4'     # minimum version
```

2. Install a list of requirements specified in a file. See the *Requirements files*.

```
$ pip install -r requirements.txt
```

3. Upgrade an already installed *SomePackage* to the latest from PyPI.

```
$ pip install --upgrade SomePackage
```

4. Install a local project in "editable" mode. See the section on *Editable Installs*.

```
$ pip install -e .                     # project in current directory
$ pip install -e path/to/project       # project in another directory
```

5. Install a project from VCS in "editable" mode. See the sections on *VCS Support* and *Editable Installs*.

```
$ pip install -e git+https://git.repo/some_pkg.git#egg=SomePackage
↪  # from git
$ pip install -e hg+https://hg.repo/some_pkg.git#egg=SomePackage
↪  # from mercurial
$ pip install -e svn+svn://svn.repo/some_pkg/trunk/#egg=SomePackage
↪  # from svn
$ pip install -e git+https://git.repo/some_pkg.git@feature
↪#egg=SomePackage  # from 'feature' branch
$ pip install -e "git+https://git.repo/some_repo.git#egg=subdir&
↪subdirectory=subdir_path" # install a python package from a repo
↪subdirectory
```

6. Install a package with setuptools extras.

```
$ pip install SomePackage[PDF]
$ pip install git+https://git.repo/some_pkg.git#egg=SomePackage[PDF]
$ pip install SomePackage[PDF]==3.0
$ pip install -e .[PDF]==3.0  # editable project in current directory
$ pip install SomePackage[PDF,EPUB]  # multiple extras
```

7. Install a particular source archive file.

```
$ pip install ./downloads/SomePackage-1.0.4.tar.gz
$ pip install http://my.package.repo/SomePackage-1.0.4.zip
```

8. Install from alternative package repositories.

   Install from a different index, and not PyPI

   ```
   $ pip install --index-url http://my.package.repo/simple/ SomePackage
   ```

   Search an additional index during install, in addition to PyPI

   ```
   $ pip install --extra-index-url http://my.package.repo/simple SomePackage
   ```

   Install from a local flat directory containing archives (and don't scan indexes):

   ```
   $ pip install --no-index --find-links=file:///local/dir/ SomePackage
   $ pip install --no-index --find-links=/local/dir/ SomePackage
   $ pip install --no-index --find-links=relative/dir/ SomePackage
   ```

9. Find pre-release and development versions, in addition to stable versions. By default, pip only finds stable versions.

   ```
   $ pip install --pre SomePackage
   ```

4.3 pip download

Contents

4.3.1 Usage

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

4.3.2 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.

`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.

4.3.3 Options

-c, --constraint <file>
: Constrain versions using the given constraints file. This option can be used multiple times.

-r, --requirement <file>
: Install from the given requirements file. This option can be used multiple times.

-b, --build <dir>
: Directory to unpack packages into and build in. Note that an initial build still takes place in a temporary directory. The location of temporary directories can be controlled by setting the TMPDIR environment variable (TEMP on Windows) appropriately.

--no-deps
: Don't install package dependencies).

--global-option <options>
: Extra global options to be supplied to the setup.py call before the install command.

--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, or one or more package names with commas between them. Note that some packages are tricky to compile and may fail to install when this option is used on them.

--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.

--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".

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

--no-clean
Don't clean up build directories).

--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.

--progress-bar <progress_bar>
Specify type of progress to be displayed [on|asciiloff|pretty|emoji] (default: on)

--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.

-d, --dest <dir>
Download packages into <dir>.

--platform <platform>
Only download wheels compatible with <platform>. Defaults to the platform of the running system.

--python-version <python_version>
Only download wheels compatible with Python interpreter version <version>. If not specified, then the current system interpreter minor version is used. A major version (e.g. '2') can be specified to match all minor revs of that major version. A minor version (e.g. '34') can also be specified.

--implementation <implementation>
Only download 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.

--abi <abi>
Only download wheels compatible with Python abi <abi>, e.g. 'pypy_41'. If not specified, then the current interpreter abi tag is used. Generally you will need to specify –implementation, –platform, and –python-version when using this option.

-i, --index-url <url>
Base URL of Python Package Index (default https://pypi.python.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.

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

--no-index
Ignore package index (only looking at –find-links URLs instead).

-f, --find-links <url>
If a url or path to an html file, then parse for links to archives. If a local path or file:// url that's a directory, then look for archives in the directory listing.

--process-dependency-links
Enable the processing of dependency links.

4.3.4 Examples

1. Download a package and all of its dependencies

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

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`).

```
$ 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.

```
$ 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.

```
$ 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.

```
$ 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
```

4.4 pip uninstall

Contents

4.4.1 Usage

```
pip [options] <package> ...
pip [options] -r <requirements file> ...
```

4.4.2 Description

Uninstall packages.

pip is able to uninstall most installed packages. Known exceptions are:

- Pure distutils packages installed with `python setup.py install`, which leave behind no metadata to determine what files were installed.
- Script wrappers installed by `python setup.py develop`.

4.4.3 Options

`-r, --requirement` `<file>`
Uninstall all the packages listed in the given requirements file. This option can be used multiple times.

`-y, --yes`
Don't ask for confirmation of uninstall deletions.

4.4.4 Examples

1. Uninstall a package.

```
$ pip uninstall simplejson
Uninstalling simplejson:
  /home/me/env/lib/python2.7/site-packages/simplejson
  /home/me/env/lib/python2.7/site-packages/simplejson-2.2.1-py2.7.egg-info
Proceed (y/n)? y
  Successfully uninstalled simplejson
```

4.5 pip freeze

Contents

4.5.1 Usage

```
pip [options]
```

4.5.2 Description

Output installed packages in requirements format.

packages are listed in a case-insensitive sorted order.

4.5.3 Options

-r, --requirement <file>
: Use the order in the given requirements file and its comments when generating output. This option can be used multiple times.

-f, --find-links <url>
: URL for finding packages, which will be added to the output.

-l, --local
: If in a virtualenv that has global access, do not output globally-installed packages.

--user
: Only output packages installed in user-site.

--all
: Do not skip these packages in the output: wheel, pip, distribute, setuptools

--exclude-editable
: Exclude editable package from output.

4.5.4 Examples

1. Generate output suitable for a requirements file.

```
$ pip freeze
docutils==0.11
Jinja2==2.7.2
MarkupSafe==0.19
Pygments==1.6
Sphinx==1.2.2
```

2. Generate a requirements file and then install from it in another environment.

```
$ env1/bin/pip freeze > requirements.txt
$ env2/bin/pip install -r requirements.txt
```

4.6 pip list

Contents

4.6.1 Usage

```
pip [options]
```

4.6.2 Description

List installed packages, including editables.

Packages are listed in a case-insensitive sorted order.

4.6.3 Options

-o, --outdated
List outdated packages

-u, --uptodate
List uptodate packages

-e, --editable
List editable projects.

-l, --local
If in a virtualenv that has global access, do not list globally-installed packages.

--user
Only output packages installed in user-site.

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

--format <list_format>
Select the output format among: columns (default), freeze, json, or legacy.

--not-required
List packages that are not dependencies of installed packages.

--exclude-editable
Exclude editable package from output.

--include-editable
Include editable package from output.

-i, --index-url <url>
Base URL of Python Package Index (default https://pypi.python.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.

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

--no-index
Ignore package index (only looking at –find-links URLs instead).

-f, --find-links <url>
If a url or path to an html file, then parse for links to archives. If a local path or file:// url that's a directory, then look for archives in the directory listing.

--process-dependency-links
Enable the processing of dependency links.

4.6.4 Examples

1. List installed packages.

```
$ pip list
docutils (0.10)
Jinja2 (2.7.2)
MarkupSafe (0.18)
Pygments (1.6)
Sphinx (1.2.1)
```

2. List outdated packages (excluding editables), and the latest version available.

```
$ pip list --outdated
docutils (Current: 0.10 Latest: 0.11)
Sphinx (Current: 1.2.1 Latest: 1.2.2)
```

3. List installed packages with column formatting.

```
$ pip list --format columns
Package Version
------- -------
docopt  0.6.2
idlex   1.13
jedi    0.9.0
```

4. List outdated packages with column formatting.

```
$ pip list -o --format columns
Package    Version Latest Type
---------- ------- ------ -----
retry      0.8.1   0.9.1  wheel
setuptools 20.6.7  21.0.0 wheel
```

5. List packages that are not dependencies of other packages. Can be combined with other options.

```
$ pip list --outdated --not-required
docutils (Current: 0.10 Latest: 0.11)
```

6. Use legacy formatting

```
$ pip list --format=legacy
colorama (0.3.7)
docopt (0.6.2)
idlex (1.13)
jedi (0.9.0)
```

7. Use json formatting

```
$ pip list --format=json
[{'name': 'colorama', 'version': '0.3.7'}, {'name': 'docopt', 'version':
↪'0.6.2'}, ...
```

8. Use freeze formatting

```
$ pip list --format=freeze
colorama==0.3.7
docopt==0.6.2
idlex==1.13
jedi==0.9.0
```

4.7 pip show

Contents

4.7.1 Usage

```
pip [options] <package> ...
```

4.7.2 Description

Show information about one or more installed packages.

4.7.3 Options

-f, --files
Show the full list of installed files for each package.

4.7.4 Examples

1. Show information about a package:

```
$ pip show sphinx
Name: Sphinx
Version: 1.4.5
Summary: Python documentation generator
Home-page: http://sphinx-doc.org/
Author: Georg Brandl
Author-email: georg@python.org
License: BSD
Location: /my/env/lib/python2.7/site-packages
Requires: docutils, snowballstemmer, alabaster, Pygments, imagesize,␣
↪Jinja2, babel, six
```

2. Show all information about a package

```
$ pip show --verbose sphinx
Name: Sphinx
Version: 1.4.5
Summary: Python documentation generator
Home-page: http://sphinx-doc.org/
Author: Georg Brandl
Author-email: georg@python.org
License: BSD
Location: /my/env/lib/python2.7/site-packages
Requires: docutils, snowballstemmer, alabaster, Pygments, imagesize,␣
↪Jinja2, babel, six
Metadata-Version: 2.0
Installer:
Classifiers:
  Development Status :: 5 - Production/Stable
  Environment :: Console
  Environment :: Web Environment
  Intended Audience :: Developers
  Intended Audience :: Education
  License :: OSI Approved :: BSD License
  Operating System :: OS Independent
  Programming Language :: Python
  Programming Language :: Python :: 2
  Programming Language :: Python :: 3
  Framework :: Sphinx
  Framework :: Sphinx :: Extension
  Framework :: Sphinx :: Theme
  Topic :: Documentation
  Topic :: Documentation :: Sphinx
  Topic :: Text Processing
  Topic :: Utilities
Entry-points:
  [console_scripts]
  sphinx-apidoc = sphinx.apidoc:main
```

```
sphinx-autogen = sphinx.ext.autosummary.generate:main
sphinx-build = sphinx:main
sphinx-quickstart = sphinx.quickstart:main
[distutils.commands]
build_sphinx = sphinx.setup_command:BuildDoc
```

4.8 pip search

Contents

4.8.1 Usage

```
pip [options] <query>
```

4.8.2 Description

Search for PyPI packages whose name or summary contains <query>.

4.8.3 Options

-i, --index <url>
Base URL of Python Package Index (default https://pypi.python.org/pypi)

4.8.4 Examples

1. Search for "peppercorn"

```
$ pip search peppercorn
pepperedform    - Helpers for using peppercorn with formprocess.
peppercorn      - A library for converting a token stream into [...]
```

4.9 pip check

Contents

4.9.1 Usage

```
pip [options]
```

4.9.2 Description

Verify installed packages have compatible dependencies.

4.9.3 Examples

1. If all dependencies are compatible:

```
$ pip check
No broken requirements found.
$ echo $?
0
```

2. If a package is missing:

```
$ pip check
pyramid 1.5.2 requires WebOb, which is not installed.
$ echo $?
1
```

3. If a package has the wrong version:

```
$ pip check
pyramid 1.5.2 has requirement WebOb>=1.3.1, but you have WebOb 0.8.
$ echo $?
1
```

4.10 pip config

Contents

4.10.1 Usage

```
pip [<file-option>] list
pip [<file-option>] [--editor <editor-path>] edit

pip [<file-option>] get name
pip [<file-option>] set name value
pip [<file-option>] unset name
```

4.10.2 Description

Manage local and global configuration.

4.10.3 Options

--editor <editor>
: Editor to use to edit the file. Uses VISUAL or EDITOR environment variables if not provided.

--global
: Use the system-wide configuration file only

--user
: Use the user configuration file only

--venv
: Use the virtualenv configuration file only

4.11 pip wheel

Contents

4.11.1 Usage

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

4.11.2 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/

Requirements: setuptools>=0.8, and wheel.

'pip wheel' uses the bdist_wheel setuptools extension from the wheel package to build individual wheels.

Build System Interface

In order for pip to build a wheel, `setup.py` must implement the `bdist_wheel` command with the following syntax:

```
python setup.py bdist_wheel -d TARGET
```

This command must create a wheel compatible with the invoking Python interpreter, and save that wheel in the directory TARGET.

No other build system commands are invoked by the `pip wheel` command.

Customising the build

It is possible using `--global-option` to include additional build commands with their arguments in the `setup.py` command. This is currently the only way to influence the building of C extensions from the command line. For example:

```
pip wheel --global-option bdist_ext --global-option -DFOO wheel
```

will result in a build command of

```
setup.py bdist_ext -DFOO bdist_wheel -d TARGET
```

which passes a preprocessor symbol to the extension build.

Such usage is considered highly build-system specific and more an accident of the current implementation than a supported interface.

4.11.3 Options

-w, --wheel-dir <dir>
: Build wheels into <dir>, where the default is the current working directory.

--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, or one or more package names with commas between them. Note that some packages are tricky to compile and may fail to install when this option is used on them.

--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.

--build-option <options>
Extra arguments to be supplied to 'setup.py bdist_wheel'.

--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.

-c, --constraint <file>
Constrain versions using the given constraints file. This option can be used multiple times.

-e, --editable <path/url>
Install a project in editable mode (i.e. setuptools "develop mode") from a local project path or a VCS url.

-r, --requirement <file>
Install from the given requirements file. This option can be used multiple times.

--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".

--ignore-requires-python
Ignore the Requires-Python information.

--no-deps
Don't install package dependencies).

-b, --build <dir>
Directory to unpack packages into and build in. Note that an initial build still takes place in a temporary directory. The location of temporary directories can be controlled by setting the TMPDIR environment variable (TEMP on Windows) appropriately.

--progress-bar <progress_bar>
Specify type of progress to be displayed [on|asciilofflprettylemoji] (default: on)

--global-option <options>
Extra global options to be supplied to the setup.py call before the 'bdist_wheel' command.

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

--no-clean
Don't clean up build directories).

--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.

-i, --index-url <url>
Base URL of Python Package Index (default https://pypi.python.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.

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

--no-index
Ignore package index (only looking at –find-links URLs instead).

-f, --find-links <url>
If a url or path to an html file, then parse for links to archives. If a local path or file:// url that's a directory, then look for archives in the directory listing.

--process-dependency-links
Enable the processing of dependency links.

4.11.4 Examples

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

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

4.12 pip hash

Contents

4.12.1 Usage

```
pip [options] <file> ...
```

4.12.2 Description

Compute a hash of a local package archive.

These can be used with –hash in a requirements file to do repeatable installs.

Overview

`pip hash` is a convenient way to get a hash digest for use with *Hash-Checking Mode*, especially for packages with multiple archives. The error message from `pip install --require-hashes ...` will give you one hash, but, if there are multiple archives (like source and binary ones), you will need to manually download and compute

a hash for the others. Otherwise, a spurious hash mismatch could occur when *pip install* is passed a different set of options, like *–no-binary*.

4.12.3 Options

-a, --algorithm <algorithm>
The hash algorithm to use: one of sha256, sha384, sha512

4.12.4 Example

Compute the hash of a downloaded archive:

```
$ pip download SomePackage
    Collecting SomePackage
      Downloading SomePackage-2.2.tar.gz
      Saved ./pip_downloads/SomePackage-2.2.tar.gz
    Successfully downloaded SomePackage
$ pip hash ./pip_downloads/SomePackage-2.2.tar.gz
    ./pip_downloads/SomePackage-2.2.tar.gz:
    --hash=sha256:93e62e05c7ad3da1a233def6731e8285156701e3419a5fe279017c429ec67ce0
```

CHAPTER 5

Development

5.1 Pull Requests

- Submit Pull Requests against the *master* branch.
- Provide a good description of what you're doing and why.
- Provide tests that cover your changes and try to run the tests locally first.

Example. Assuming you set up GitHub account, forked pip repository from https://github.com/pypa/pip to your own page via web interface, and your fork is located at https://github.com/yourname/pip

```
$ git clone git@github.com:pypa/pip.git
$ cd pip
# ...
$ git diff
$ git add <modified> ...
$ git status
$ git commit
```

You may reference relevant issues in commit messages (like #1259) to make GitHub link issues and commits together, and with phrase like "fixes #1259" you can even close relevant issues automatically. Now push the changes to your fork:

```
$ git push git@github.com:yourname/pip.git
```

Open Pull Requests page at https://github.com/yourname/pip/pulls and click "New pull request". That's it.

5.2 Automated Testing

All pull requests and merges to 'master' branch are tested in Travis based on our .travis.yml file.

Usually, a link to your specific travis build appears in pull requests, but if not, you can find it on our travis pull requests page

The only way to trigger Travis to run again for a pull request, is to submit another change to the pull branch.

We also have Jenkins CI that runs regularly for certain python versions on windows and centos.

5.3 Running tests

OS Requirements: subversion, bazaar, git, and mercurial.

Python Requirements: tox or pytest, virtualenv, scripttest, and mock

Ways to run the tests locally:

```
$ tox -e py33           # The preferred way to run the tests, can use pyNN to
                        # run for a particular version or leave off the -e to
                        # run for all versions.
$ python setup.py test  # Using the setuptools test plugin
$ py.test               # Using py.test directly
$ tox                   # Using tox against pip's tox.ini
```

If you are missing one of the VCS tools, you can tell `py.test` to skip it:

```
$ py.test -k 'not bzr'
$ py.test -k 'not svn'
```

5.4 Getting Involved

The pip project welcomes help in the following ways:

- Making Pull Requests for code, tests, or docs.
- Commenting on open issues and pull requests.
- Helping to answer questions on the mailing list.

If you want to become an official maintainer, start by helping out.

Later, when you think you're ready, get in touch with one of the maintainers, and they will initiate a vote.

5.5 Adding a NEWS Entry

The `NEWS.rst` file is managed using towncrier and all non trivial changes must be accompanied by a news entry.

To add an entry to the news file, first you need to have created an issue describing the change you want to make. A Pull Request itself *may* function as such, but it is preferred to have a dedicated issue (for example, in case the PR ends up rejected due to code quality reasons).

Once you have an issue or pull request, you take the number and you create a file inside of the `news/` directory named after that issue number with an extension of `removal`, `feature`, `bugfix`, or `doc`. Thus if your issue or PR number is `1234` and this change is fixing a bug, then you would create a file `news/1234.bugfix`. PRs can span multiple categories by creating multiple files (for instance, if you added a feature and deprecated/removed the old feature at the same time, you would create `news/NNNN.feature` and `news/NNNN.removal`). Likewise if a PR touches multiple issues/PRs you may create a file for each of them with the exact same contents and Towncrier will deduplicate them.

The contents of this file are reStructuredText formatted text that will be used as the content of the news file entry. You do not need to reference the issue or PR numbers here as towncrier will automatically add a reference to all of the affected issues when rendering the news file.

A trivial change is anything that does not warrant an entry in the news file. Some examples are: Code refactors that don't change anything as far as the public is concerned, typo fixes, white space modification, etc. To mark a PR as trivial a contributor simply needs to add a randomly named, empty file to the `news/` directory with the extension of `.trivial`. If you are on a POSIX like operating system, one can be added by running `touch news/$(uuidgen).trivial`. Core committers may also add a "trivial" label to the PR which will accomplish the same thing.

Upgrading, removing, or adding a new vendored library gets a special mention using a `news/<library>.vendor` file. This is in addition to any features, bugfixes, or other kinds of news that pulling in this library may have. This uses the library name as the key so that updating the same library twice doesn't produce two news file entries.

5.6 Release Process

1. On the current pip `master` branch, generate a new `AUTHORS.txt` by running `invoke generate.authors` and commit the results.
2. On the current pip `master` branch, make a new commit which bumps the version in `pip/__init__.py` to the release version and adjust the `CHANGES.txt` file to reflect the current date.
3. On the current pip `master` branch, generate a new `NEWS.rst` by running `invoke generate.news` and commit the results.
4. Create a signed tag of the `master` branch of the form `X.Y.Z` using the command `git tag -s X.Y.Z`.
5. Checkout the tag using `git checkout X.Y.Z` and create the distribution files using `python setup.py sdist bdist_wheel`.
6. Upload the distribution files to PyPI using twine (`twine upload -s dist/*`). The upload should include GPG signatures of the distribution files.
7. Push all of the changes.
8. Regenerate the `get-pip.py` script by running `invoke generate.installer` in the get-pip repository, and committing the results.

5.7 Creating a Bugfix Release

Sometimes we need to release a bugfix release of the form `X.Y.Z+1`. In order to create one of these the changes should already be merged into the `master` branch.

1. Create a new `release/X.Y.Z+1` branch off of the `X.Y.Z` tag using the command `git checkout -b release/X.Y.Z+1 X.Y.Z`.
2. Cherry pick the fixed commits off of the `master` branch, fixing any conflicts and moving any changelog entries from the development version's changelog section to the `X.Y.Z+1` section.
3. Push the `release/X.Y.Z+1` branch to github and submit a PR for it against the `master` branch and wait for the tests to run.
4. Once tests run, merge the `release/X.Y.Z+1` branch into master, and follow the above release process starting with step 4.

CHAPTER 6

Release Notes

6.1 9.0.3 (2018-03-21)

- Fix an error where the vendored requests was not correctly containing itself to only the internal vendored prefix.
- Restore compatability with 2.6.

6.2 9.0.2 (2018-03-16)

- Fallback to using SecureTransport on macOS when the linked OpenSSL is too old to support TLSv1.2.

6.3 9.0.1 (2016-11-06)

- Correct the deprecation message when not specifying a –format so that it uses the correct setting name (`format`) rather than the incorrect one (`list_format`). (#4058)
- Fix `pip check` to check all available distributions and not just the local ones. (#4083)
- Fix a crash on non ASCII characters from *lsb_release*. (#4062)
- Fix an SyntaxError in an unused module of a vendored dependency. (#4059)
- Fix UNC paths on Windows. (#4064)

6.4 9.0.0 (2016-11-02)

- **BACKWARD INCOMPATIBLE** Remove the attempted autodetection of requirement names from URLs, URLs must include a name via `#egg=`.

- **DEPRECATION** `pip install --egg` have been deprecated and will be removed in the future. This "feature" has a long list of drawbacks which break nearly all of pip's other features in subtle and hard-to-diagnose ways.
- **DEPRECATION** `--default-vcs` option. (#4052)
- **WARNING** pip 9 cache can break forward compatibility with previous pip versions if your package repository allows chunked responses. (#4078)
- Add an `--upgrade-strategy` option to `pip install`, to control how dependency upgrades are managed. (#3972)
- Add a `pip check` command to check installed packages dependencies. (#3750)
- Add option allowing user to abort pip operation if file/directory exists
- Add Appveyor CI
- Uninstall existing packages when performing an editable installation of the same packages. (#1548)
- `pip show` is less verbose by default. `--verbose` prints multiline fields. (#3858)
- Add optional column formatting to `pip list`. (#3651)
- Add `--not-required` option to `pip list`, which lists packages that are not dependencies of other packages.
- Fix builds on systems with symlinked `/tmp` directory for custom builds such as numpy. (#3701)
- Fix regression in `pip freeze`: when there is more than one git remote, priority is given to the remote named `origin`. (#3708, #3616).
- Fix crash when calling `pip freeze` with invalid requirement installed. (#3704, #3681)
- Allow multiple `--requirement` files in `pip freeze`. (#3703)
- Implementation of pep-503 `data-requires-python`. When this field is present for a release link, pip will ignore the download when installing to a Python version that doesn't satisfy the requirement.
- `pip wheel` now works on editable packages too (it was only working on editable dependencies before); this allows running `pip wheel` on the result of `pip freeze` in presence of editable requirements. (#3695, #3291)
- Load credentials from `.netrc` files. (#3715, #3569)
- Add `--platform`, `--python-version`, `--implementation` and `--abi` parameters to `pip download`. These allow utilities and advanced users to gather distributions for interpreters other than the one pip is being run on. (#3760)
- Skip scanning virtual environments, even when venv/bin/python is a dangling symlink.
- Added `pip completion` support for the `fish` shell.
- Fix problems on Windows on Python 2 when username or hostname contains non-ASCII characters. (#3463, #3970, #4000)
- Use `git fetch --tags` to fetch tags in addition to everything else that is normally fetched; this is necessary in case a git requirement url points to a tag or commit that is not on a branch. (#3791)
- Normalize package names before using in `pip show` (#3976)
- Raise when Requires-Python do not match the running version and add `--ignore-requires-python` option as escape hatch. (#3846)
- Report the correct installed version when performing an upgrade in some corner cases. (#2382
- Add `-i` shorthand for `--index` flag in `pip search`.

- Do not optionally load C dependencies in requests. (#1840, #2930, #3024)
- Strip authentication from SVN url prior to passing it to `svn`. (#3697, #3209)
- Also install in platlib with `--target` option. (#3694, #3682)
- Restore the ability to use inline comments in requirements files passed to `pip freeze`. (#3680)

6.5 8.1.2 (2016-05-10)

- Fix a regression on systems with uninitialized locale. (#3575)
- Use environment markers to filter packages before determining if a required wheel is supported. (#3254)
- Make glibc parsing for *manylinux1* support more robust for the variety of glibc versions found in the wild. (#3588)
- Update environment marker support to fully support legacy and PEP 508 style environment markers. (#3624)
- Always use debug logging to the `--log` file. (#3351)
- Don't attempt to wrap search results for extremely narrow terminal windows. (#3655)

6.6 8.1.1 (2016-03-17)

- Fix regression with non-ascii requirement files on Python 2 and add support for encoding headers in requirement files. (#3548, #3547)

6.7 8.1.0 (2016-03-05)

- Implement PEP 513, which adds support for the manylinux1 platform tag, allowing carefully compiled binary wheels to be installed on compatible Linux platforms.
- Allow wheels which are not specific to a particular Python interpreter but which are specific to a particular platform. (#3202)
- Fixed an issue where `call_subprocess` would crash trying to print debug data on child process failure. (#3521, #3522)
- Exclude the wheel package from the *pip freeze* output (like pip and setuptools). (#2989)
- Allow installing modules from a subdirectory of a vcs repository in non-editable mode. (#3217, #3466)
- Make pip wheel and pip download work with vcs urls with subdirectory option. (#3466)
- Show classifiers in `pip show`.
- Show PEP376 Installer in `pip show`. (#3517)
- Unhide completion command. (#1810)
- Show latest version number in `pip search` results. (#1415)
- Decode requirement files according to their BOM if present. (#3485, #2865)
- Fix and deprecate package name detection from url path. (#3523, #3495)
- Correct the behavior where interpreter specific tags (such as cp34) were being used on later versions of the same interpreter instead of only for that specific interpreter. (#3472)

- Fix an issue where pip would erroneously install a 64 bit wheel on a 32 bit Python running on a 64 bit macOS machine.
- Do not assume that all git repositories have an origin remote.
- Correctly display the line to add to a requirements.txt for an URL based dependency when `--require-hashes` is enabled.

6.8 8.0.3 (2016-02-25)

- Make `install --quiet` really quiet. (#3418)
- Fix a bug when removing packages in python 3: disable INI-style parsing of the entry_point.txt file to allow entry point names with colons. (#3434)
- Normalize generated script files path in RECORD files. (#3448)
- Fix bug introduced in 8.0.0 where subcommand output was not shown, even when the user specified `-v` / `--verbose`. (#3486)
- Enable python -W with respect to PipDeprecationWarning. (#3455)
- Upgrade distlib to 0.2.2.
- Improved support for Jython when quoting executables in output scripts. (#3467)
- Add a *–all* option to *pip freeze* to include usually skipped package (like pip, setuptools and wheel) to the freeze output. (#1610)

6.9 8.0.2 (2016-01-21)

- Stop attempting to trust the system CA trust store because it's extremely common for them to be broken, often in incompatible ways. (#3416)

6.10 8.0.1 (2016-01-21)

- Detect CAPaths in addition to CAFiles on platforms that provide them.
- Installing argparse or wsgiref will no longer warn or error - pip will allow the installation even though it may be useless (since the installed thing will be shadowed by the standard library).
- Upgrading a distutils installed item that is installed outside of a virtual environment, while inside of a virtual environment will no longer warn or error.
- Fix a bug where pre-releases were showing up in `pip list --outdated` without the `--pre` flag.
- Switch the SOABI emulation from using RuntimeWarnings to debug logging.
- Rollback the removal of the ability to uninstall distutils installed items until a future date.

6.11 8.0.0 (2016-01-19)

- **BACKWARD INCOMPATIBLE** Drop support for Python 3.2.

- **BACKWARD INCOMPATIBLE** Remove the ability to find any files other than the ones directly linked from the index or find-links pages.
- **BACKWARD INCOMPATIBLE** Remove the `--download-cache` which had been deprecated and no-op'd in 6.0.
- **BACKWARD INCOMPATIBLE** Remove the `--log-explicit-levels` which had been deprecated in 6.0.
- **BACKWARD INCOMPATIBLE** Change pip wheel –wheel-dir default path from <cwd>/wheelhouse to <cwd>.
- Deprecate and no-op the `--allow-external`, `--allow-all-external`, and `--allow-unverified` functionality that was added as part of PEP 438. With changes made to the repository protocol made in PEP 470, these options are no longer functional.
- Allow `--trusted-host` within a requirements file. (#2822)
- Allow `--process-dependency-links` within a requirements file. (#1274)
- Allow `--pre` within a requirements file. (#1273)
- Allow repository URLs with secure transports to count as trusted. (E.g., "git+ssh" is okay.) (#2811)
- Implement a top-level `pip download` command and deprecate `pip install --download`.
- When uninstalling, look for the case of paths containing symlinked directories (#3141, #3154)
- When installing, if building a wheel fails, clear up the build directory before falling back to a source install. (#3047)
- Fix user directory expansion when `HOME=/`. Workaround for Python bug http://bugs.python.org/issue14768. (#2996)
- Correct reporting of requirements file line numbers. (#3009, #3125)
- Fixed Exception(IOError) for `pip freeze` and `pip list` commands with subversion >= 1.7. (#1062, #3346)
- Provide a spinner showing that progress is happening when installing or building a package via `setup.py`. This will alleviate concerns that projects with unusually long build times have with pip appearing to stall.
- Include the functionality of `peep` into pip, allowing hashes to be baked into a requirements file and ensuring that the packages being downloaded match one of those hashes. This is an additional, opt-in security measure that, when used, removes the need to trust the repository.
- Fix a bug causing pip to not select a wheel compiled against an OSX SDK later than what Python itself was compiled against when running on a newer version of OSX.
- Add a new `--prefix` option for `pip install` that supports wheels and sdists. (#3252)
- Fixed issue regarding wheel building with setup.py using a different encoding than the system. (#2042)
- Drop PasteScript specific egg_info hack. (#3270)
- Allow combination of pip list options –editable with –outdated/–uptodate. (#933)
- Gives VCS implementations control over saying whether a project is under their control. (#3258)
- Git detection now works when `setup.py` is not at the Git repo root and when `package_dir` is used, so `pip freeze` works in more cases. (#3258)
- Correctly freeze Git develop packages in presence of the &subdirectory option (#3258)
- The detection of editable packages now relies on the presence of `.egg-link` instead of looking for a VCS, so `pip list -e` is more reliable. (#3258)

- Add the `--prefix` flag to `pip install` which allows specifying a root prefix to use instead of `sys.prefix`. (#3252)
- Allow duplicate specifications in the case that only the extras differ, and union all specified extras together. (#3198)
- Fix the detection of the user's current platform on OSX when determining the OSX SDK version. (#3232)
- Prevent the automatically built wheels from mistakenly being used across multiple versions of Python when they may not be correctly configured for that by making the wheel specific to a specific version of Python and specific interpreter. (#3225)
- Emulate the SOABI support in wheels from Python 2.x on Python 2.x as closely as we can with the information available within the interpreter. (#3075)
- Don't roundtrip to the network when git is pinned to a specific commit hash and that hash already exists locally. (#3066)
- Prefer wheels built against a newer SDK to wheels built against an older SDK on OSX. (#3163)
- Show entry points for projects installed via wheel. (#3122)
- Improve message when an unexisting path is passed to –find-links option. (#2968)
- pip freeze does not add the VCS branch/tag name in the #egg=... fragment anymore. (#3312)
- Warn on installation of editable if the provided #egg=name part does not match the metadata produced by *setup.py egg_info*. (#3143)
- Add support for .xz files for python versions supporting them (>= 3.3). (#722)

6.12 7.1.2 (2015-08-22)

- Don't raise an error if pip is not installed when checking for the latest pip version.

6.13 7.1.1 (2015-08-20)

- Check that the wheel cache directory is writable before we attempt to write cached files to them.
- Move the pip version check until *after* any installs have been performed, thus removing the extraneous warning when upgrading pip.
- Added debug logging when using a cached wheel.
- Respect platlib by default on platforms that have it separated from purelib.
- Upgrade packaging to 15.3. - Normalize post-release spellings for rev/r prefixes.
- Upgrade distlib to 0.2.1. - Updated launchers to decode shebangs using UTF-8. This allows non-ASCII pathnames to be correctly handled. - Ensured that the executable written to shebangs is normcased. - Changed ScriptMaker to work better under Jython.
- Upgrade ipaddress to 1.0.13.

6.14 7.1.0 (2015-06-30)

- Allow constraining versions globally without having to know exactly what will be installed by the pip command. (#2731)

- Accept –no-binary and –only-binary via pip.conf. (#2867)
- Allow `--allow-all-external` within a requirements file.
- Fixed an issue where `--user` could not be used when `--prefix` was used in a distutils configuration file.
- Fixed an issue where the SOABI tags were not correctly being generated on Python 3.5.
- Fixed an issue where we were advising windows users to upgrade by directly executing pip, when that would always fail on Windows.
- Allow ~ to be expanded within a cache directory in all situations.

6.15 7.0.3 (2015-06-01)

- Fixed a regression where `--no-cache-dir` would raise an exception. (#2855)

6.16 7.0.2 (2015-06-01)

- **BACKWARD INCOMPATIBLE** Revert the change (released in v7.0.0) that required quoting in requirements files around specifiers containing environment markers. (#2841)
- **BACKWARD INCOMPATIBLE** Revert the accidental introduction of support for options interleaved with requirements, version specifiers etc in `requirements` files. (#2841)
- Expand ~ in the cache directory when caching wheels. (#2816)
- Use `python -m pip` instead of `pip` when recommending an upgrade command to Windows users.

6.17 7.0.1 (2015-05-22)

- Don't build and cache wheels for non-editable installations from VCSs.
- Allow `--allow-all-external` inside of a requirements.txt file, fixing a regression in 7.0.

6.18 7.0.0 (2015-05-21)

- **BACKWARD INCOMPATIBLE** Removed the deprecated `--mirror`, `--use-mirrors`, and `-M` options.
- **BACKWARD INCOMPATIBLE** Removed the deprecated `zip` and `unzip` commands.
- **BACKWARD INCOMPATIBLE** Removed the deprecated `--no-install` and `--no-download` options.
- **BACKWARD INCOMPATIBLE** No longer implicitly support an insecure origin origin, and instead require insecure origins be explicitly trusted with the `--trusted-host` option.
- **BACKWARD INCOMPATIBLE** Removed the deprecated link scraping that attempted to parse HTML comments for a specially formatted comment.
- **BACKWARD INCOMPATIBLE** Requirements in requirements files containing markers must now be quoted due to parser changes. For example, use `"SomeProject; python_version < '2.7'"`, not simply `SomeProject; python_version < '2.7'` (#2697, #2725)
- *get-pip.py* now installs the "wheel" package, when it's not already installed. (#2800)

- Ignores bz2 archives if Python wasn't compiled with bz2 support. (#497)
- Support `--install-option` and `--global-option` per requirement in requirement files. (#2537)
- Build Wheels prior to installing from sdist, caching them in the pip cache directory to speed up subsequent installs. (#2618)
- Allow fine grained control over the use of wheels and source builds. (#2699)
- `--no-use-wheel` and `--use-wheel` are deprecated in favour of new options `--no-binary` and `--only-binary`. The equivalent of `--no-use-wheel` is `--no-binary=:all:`. (#2699)
- The use of `--install-option`, `--global-option` or `--build-option` disable the use of wheels, and the autobuilding of wheels. (#2711, #2677)
- Improve logging when a requirement marker doesn't match your environment. (#2735)
- Removed the temporary modifications (that began in pip v1.4 when distribute and setuptools merged) that allowed distribute to be considered a conflict to setuptools. `pip install -U setuptools` will no longer upgrade "distribute" to "setuptools". Instead, use `pip install -U distribute`. (#2767)
- Only display a warning to upgrade pip when the newest version is a final release and it is not a post release of the version we already have installed. (#2766)
- Display a warning when attempting to access a repository that uses HTTPS when we don't have Python compiled with SSL support. (#2761)
- Allowing using extras when installing from a file path without requiring the use of an editable. (#2785)
- Fix an infinite loop when the cache directory is stored on a file system which does not support hard links. (#2796)
- Remove the implicit debug log that was written on every invocation, instead users will need to use `--log` if they wish to have one. (#2798)

6.19 6.1.1 (2015-04-07)

- No longer ignore dependencies which have been added to the standard library, instead continue to install them.

6.20 6.1.0 (2015-04-07)

- Fixes upgrades failing when no potential links were found for dependencies other than the current installation. (#2538, #2502)
- Use a smoother progress bar when the terminal is capable of handling it, otherwise fallback to the original ASCII based progress bar.
- Display much less output when *pip install* succeeds, because on success, users probably don't care about all the nitty gritty details of compiling and installing. When *pip install* fails, display the failed install output once instead of twice, because once is enough. (#2487)
- Upgrade the bundled copy of requests to 2.6.0, fixing CVE-2015-2296.
- Display format of latest package when using `pip list --outdated`. (#2475)
- Don't use pywin32 as ctypes should always be available on Windows, using pywin32 prevented uninstallation of pywin32 on Windows. (PR #2467)
- Normalize the `--wheel-dir` option, expanding out constructs such as ~ when used. (#2441)

- Display a warning when an undefined extra has been requested. (#2142)
- Speed up installing a directory in certain cases by creating a sdist instead of copying the entire directory. (#2535)
- Don't follow symlinks when uninstalling files (#2552)
- Upgrade the bundled copy of cachecontrol from 0.11.1 to 0.11.2. (#2481, #2595)
- Attempt to more smartly choose the order of installation to try and install dependencies before the projects that depend on them. (#2616)
- Skip trying to install libraries which are part of the standard library. (#2636, #2602)
- Support arch specific wheels that are not tied to a specific Python ABI. (#2561)
- Output warnings and errors to stderr instead of stdout. (#2543)
- Adjust the cache dir file checks to only check ownership if the effective user is root. (#2396)
- Install headers into a per project name directory instead of all of them into the root directory when inside of a virtual environment. (#2421)

6.21 6.0.8 (2015-02-04)

- Fix an issue where the `--download` flag would cause pip to no longer use randomized build directories.
- Fix an issue where pip did not properly unquote quoted URLs which contain characters like PEP 440's epoch separator (`!`).
- Fix an issue where distutils installed projects were not actually uninstalled and deprecate attempting to uninstall them altogether.
- Retry deleting directories in case a process like an antivirus is holding the directory open temporarily.
- Fix an issue where pip would hide the cursor on Windows but would not reshow it.

6.22 6.0.7 (2015-01-28)

- Fix a regression where Numpy requires a build path without symlinks to properly build.
- Fix a broken log message when running `pip wheel` without a requirement.
- Don't mask network errors while downloading the file as a hash failure.
- Properly create the state file for the pip version check so it only happens once a week.
- Fix an issue where switching between Python 3 and Python 2 would evict cached items.
- Fix a regression where pip would be unable to successfully uninstall a project without a normalized version.

6.23 6.0.6 (2015-01-03)

- Continue the regression fix from 6.0.5 which was not a complete fix.

6.24 6.0.5 (2015-01-03)

- Fix a regression with 6.0.4 under Windows where most commands would raise an exception due to Windows not having the `os.geteuid()` function.

6.25 6.0.4 (2015-01-03)

- Fix an issue where ANSI escape codes would be used on Windows even though the Windows shell does not support them, causing odd characters to appear with the progress bar.
- Fix an issue where using -v would cause an exception saying `TypeError: not all arguments converted during string formatting`.
- Fix an issue where using -v with dependency links would cause an exception saying `TypeError: 'InstallationCandidate' object is not iterable`.
- Fix an issue where upgrading distribute would cause an exception saying `TypeError: expected string or buffer`.
- Show a warning and disable the use of the cache directory when the cache directory is not owned by the current user, commonly caused by using `sudo` without the `-H` flag.
- Update PEP 440 support to handle the latest changes to PEP 440, particularly the changes to `>V` and `<V` so that they no longer imply `!=V.*`.
- Document the default cache directories for each operating system.
- Create the cache directory when the pip version check needs to save to it instead of silently logging an error.
- Fix a regression where the `-q` flag would not properly suppress the display of the progress bars.

6.26 6.0.3 (2014-12-23)

- Fix an issue where the implicit version check new in pip 6.0 could cause pip to block for up to 75 seconds if PyPI was not accessible.
- Make `--no-index` imply `--disable-pip-version-check`.

6.27 6.0.2 (2014-12-23)

- Fix an issue where the output saying that a package was installed would report the old version instead of the new version during an upgrade.
- Fix left over merge conflict markers in the documentation.
- Document the backwards incompatible PEP 440 change in the 6.0.0 changelog.

6.28 6.0.1 (2014-12-22)

- Fix executable file permissions for Wheel files when using the distutils scripts option.
- Fix a confusing error message when an exceptions was raised at certain points in pip's execution.

- Fix the missing list of versions when a version cannot be found that matches the specifiers.
- Add a warning about the possibly problematic use of > when the given specifier doesn't match anything.
- Fix an issue where installing from a directory would not copy over certain directories which were being excluded, however some build systems rely on them.

6.29 6.0 (2014-12-22)

- **PROCESS** Version numbers are now simply `X.Y` where the leading `1` has been dropped.
- **BACKWARD INCOMPATIBLE** Dropped support for Python 3.1.
- **BACKWARD INCOMPATIBLE** Removed the bundle support which was deprecated in 1.4. (#1806)
- **BACKWARD INCOMPATIBLE** File lists generated by *pip show -f* are now rooted at the location reported by show, rather than one (unstated) directory lower. (#1933)
- **BACKWARD INCOMPATIBLE** The ability to install files over the FTP protocol was accidentally lost in pip 1.5 and it has now been decided to not restore that ability.
- **BACKWARD INCOMPATIBLE** PEP 440 is now fully implemented, this means that in some cases versions will sort differently or version specifiers will be interpreted differently than previously. The common cases should all function similarly to before.
- **DEPRECATION** `pip install --download-cache` and `pip wheel --download-cache` command line flags have been deprecated and the functionality removed. Since pip now automatically configures and uses it's internal HTTP cache which supplants the `--download-cache` the existing options have been made non functional but will still be accepted until their removal in pip v8.0. For more information please see https://pip.pypa.io/en/stable/reference/pip_install.html#caching
- **DEPRECATION** `pip install --build` and `pip install --no-clean` are now *NOT* deprecated. This reverses the deprecation that occurred in v1.5.3. (#906)
- **DEPRECATION** Implicitly accessing URLs which point to an origin which is not a secure origin, instead requiring an opt-in for each host using the new `--trusted-host` flag (`pip install --trusted-host example.com foo`).
- Allow the new `--trusted-host` flag to also disable TLS verification for a particular hostname.
- Added a `--user` flag to `pip freeze` and `pip list` to check the user site directory only.
- Silence byte compile errors when installation succeed. (#1873)
- Added a virtualenv-specific configuration file. (#1364)
- Added site-wide configuration files. (1978)
- Added an automatic check to warn if there is an updated version of pip available. (#2049)
- *wsgiref* and *argparse* (for >py26) are now excluded from *pip list* and *pip freeze*. (#1606, #1369)
- Add `--client-cert` option for SSL client certificates. (#1424)
- *pip show –files* was broken for wheel installs. (#1635, #1484)
- install_lib should take precedence when reading distutils config. (#1642, #1641)
- Send *Accept-Encoding: identity* when downloading files in an attempt to convince some servers who double compress the downloaded file to stop doing so. (#1688)
- Stop breaking when given pip commands in uppercase (#1559, #1725)

- Pip no longer adds duplicate logging consumers, so it won't create duplicate output when being called multiple times. (#1618, #1723)
- *pip wheel* now returns an error code if any wheels fail to build. (#1769)
- *pip wheel* wasn't building wheels for dependencies of editable requirements. (#1775)
- Allow the use of `--no-use-wheel` within a requirements file. (#1859)
- Attempt to locate system TLS certificates to use instead of the included CA Bundle if possible. (#1680, #1866)
- Allow use of Zip64 extension in Wheels and other zip files. (#1319, #1868)
- Properly handle an index or –find-links target which has a <base> without a href attribute. (#1101, #1869)
- Properly handle extras when a project is installed via Wheel. (#1885, #1896)
- Added support to respect proxies in `pip search`. (#1180, #932, #1104, #1902)
- *pip install –download* works with vcs links. (#798, #1060, #1926)
- Disabled warning about insecure index host when using localhost. Based off of Guy Rozendorn's work in #1718. (#1456, #1967)
- Allow the use of OS standard user configuration files instead of ones simply based around `$HOME`. (#2021)
- When installing directly from wheel paths or urls, previous versions were not uninstalled. (#1825, #804, #1838)
- Detect the location of the `.egg-info` directory by looking for any file located inside of it instead of relying on the record file listing a directory. (#2075, #2076)
- Use a randomized and secure default build directory when possible. (#1964, #1935, #676, #2122, CVE-2014-8991)
- Support environment markers in requirements.txt files. (#1433, #2134)
- Automatically retry failed HTTP requests by default. (#1444, #2147)
- Handle HTML Encoding better using a method that is more similar to how browsers handle it. (#1100, #1874)
- Reduce the verbosity of the pip command by default. (#2175, #2177, #2178)
- Fixed #2031 - Respect sys.executable on OSX when installing from Wheels.
- Display the entire URL of the file that is being downloaded when downloading from a non PyPI repository. (#2183)
- Support setuptools style environment markers in a source distribution. (#2153)

6.30 1.5.6 (2014-05-16)

- Upgrade requests to 2.3.0 to fix an issue with proxies on Python 3.4.1. (#1821)

6.31 1.5.5 (2014-05-03)

- Uninstall issues on debianized pypy, specifically issues with setuptools upgrades. (#1632, #1743)
- Update documentation to point at https://bootstrap.pypa.io/get-pip.py for bootstrapping pip.
- Update docs to point to https://pip.pypa.io/
- Upgrade the bundled projects (distlib==0.1.8, html5lib==1.0b3, six==1.6.1, colorama==0.3.1, setuptools==3.4.4).

6.32 1.5.4 (2014-02-21)

- Correct deprecation warning for `pip install --build` to only notify when the *–build* value is different than the default.

6.33 1.5.3 (2014-02-20)

- **DEPRECATION** `pip install --build` and `pip install --no-clean` are now deprecated. (#906)
- Fixed being unable to download directly from wheel paths/urls, and when wheel downloads did occur using requirement specifiers, dependencies weren't downloaded. (#1112, #1527)
- `pip wheel` was not downloading wheels that already existed. (#1320, #1524)
- `pip install --download` was failing using local `--find-links`. (#1111, #1524)
- Workaround for Python bug http://bugs.python.org/issue20053. (#1544)
- Don't pass a unicode __file__ to setup.py on Python 2.x. (#1583)
- Verify that the Wheel version is compatible with this pip. (#1569)

6.34 1.5.2 (2014-01-26)

- Upgraded the vendored `pkg_resources` and `_markerlib` to setuptools 2.1.
- Fixed an error that prevented accessing PyPI when pyopenssl, ndg-httpsclient, and pyasn1 are installed.
- Fixed an issue that caused trailing comments to be incorrectly included as part of the URL in a requirements file.

6.35 1.5.1 (2014-01-20)

- pip now only requires setuptools (any setuptools, not a certain version) when installing distributions from src (i.e. not from wheel). (#1434)
- *get-pip.py* now installs setuptools, when it's not already installed. (#1475)
- Don't decode downloaded files that have a `Content-Encoding` header. (#1435)
- Fix to correctly parse wheel filenames with single digit versions. (#1445)
- If *–allow-unverified* is used assume it also means *–allow-external*. (#1457)

6.36 1.5 (2014-01-01)

- **BACKWARD INCOMPATIBLE** pip no longer supports the `--use-mirrors`, `-M`, and `--mirrors` flags. The mirroring support has been removed. In order to use a mirror specify it as the primary index with `-i` or `--index-url`, or as an additional index with `--extra-index-url`. (#1098, CVE-2013-5123)

- **BACKWARD INCOMPATIBLE** pip no longer will scrape insecure external urls by default nor will it install externally hosted files by default. Users may opt into installing externally hosted or insecure files or urls using `--allow-external PROJECT` and `--allow-unverified PROJECT`. (#1055)
- **BACKWARD INCOMPATIBLE** pip no longer respects dependency links by default. Users may opt into respecting them again using `--process-dependency-links`.
- **DEPRECATION** `pip install --no-install` and `pip install --no-download` are now formally deprecated. See #906 for discussion on possible alternatives, or lack thereof, in future releases.
- **DEPRECATION** `pip zip` and `pip unzip` are now formally deprecated.
- pip will now install Mac OSX platform wheels from PyPI. (PR #1278)
- pip now generates the appropriate platform-specific console scripts when installing wheels. (#1251)
- Pip now confirms a wheel is supported when installing directly from a path or url. (#1315)
- `--ignore-installed` now behaves again as designed, after it was unintentionally broke in v0.8.3 when fixing #14. (#1097, #1352)
- Fixed a bug where global scripts were being removed when uninstalling –user installed packages. (#1353)
- `--user` wasn't being respected when installing scripts from wheels. (#1163, #1176)
- Assume '_' means '-' in versions from wheel filenames. (#1150, #1158)
- Error when using –log with a failed install. (#219, #1205)
- Fixed logging being buffered and choppy in Python 3. (#1131)
- Don't ignore –timeout. (#70, #1202)
- Fixed an error when setting PIP_EXISTS_ACTION. (#772, #1201)
- Added colors to the logging output in order to draw attention to important warnings and errors. (#1109)
- Added warnings when using an insecure index, find-link, or dependency link. (#1121)
- Added support for installing packages from a subdirectory using the `subdirectory` editable option. (#1082)
- Fixed "TypeError: bad operand type for unary" in some cases when installing wheels using –find-links. (#1192, #1218)
- Archive contents are now written based on system defaults and umask (i.e. permissions are not preserved), except that regular files with any execute permissions have the equivalent of "chmod +x" applied after being written. (#1133, #317, #1146)
- PreviousBuildDirError now returns a non-zero exit code and prevents the previous build dir from being cleaned in all cases. (#1162)
- Renamed –allow-insecure to –allow-unverified, however the old name will continue to work for a period of time. (#1257)
- Fixed an error when installing local projects with symlinks in Python 3. (#1006, #1311)
- The previously hidden `--log-file` option, is now shown as a general option. (#1316)

6.37 1.4.1 (2013-08-07)

- **New Signing Key** Release 1.4.1 is using a different key than normal with fingerprint: 7C6B 7C5D 5E2B 6356 A926 F04F 6E3C BCE9 3372 DCFA
- Fixed issues with installing from pybundle files. (#1116)

- Fixed error when sysconfig module throws an exception. (#1095)
- Don't ignore already installed pre-releases. (#1076)
- Fixes related to upgrading setuptools. (#1092)
- Fixes so that –download works with wheel archives. (#1113)
- Fixes related to recognizing and cleaning global build dirs. (#1080)

6.38 1.4 (2013-07-23)

- **BACKWARD INCOMPATIBLE** pip now only installs stable versions by default, and offers a new `--pre` option to also find pre-release and development versions. (#834)
- **BACKWARD INCOMPATIBLE** Dropped support for Python 2.5. The minimum supported Python version for pip 1.4 is Python 2.6.
- Added support for installing and building wheel archives. Thanks Daniel Holth, Marcus Smith, Paul Moore, and Michele Lacchia (#845)
- Applied security patch to pip's ssl support related to certificate DNS wildcard matching (http://bugs.python.org/issue17980).
- To satisfy pip's setuptools requirement, pip now recommends setuptools>=0.8, not distribute. setuptools and distribute are now merged into one project called 'setuptools'. (#1003)
- pip will now warn when installing a file that is either hosted externally to the index or cannot be verified with a hash. In the future pip will default to not installing them and will require the flags –allow-external NAME, and –allow-insecure NAME respectively. (#985)
- If an already-downloaded or cached file has a bad hash, re-download it rather than erroring out. (#963)
- `pip bundle` and support for installing from pybundle files is now considered deprecated and will be removed in pip v1.5.
- Fix a number of issues related to cleaning up and not reusing build directories. (#413, #709, #634, #602, #939, #865, #948)
- Added a User Agent so that pip is identifiable in logs. (#901)
- Added ssl and –user support to get-pip.py. Thanks Gabriel de Perthuis. (#895)
- Fixed the proxy support, which was broken in pip 1.3.x (#840)
- Fixed pip failing when server does not send content-type header. Thanks Hugo Lopes Tavares and Kelsey Hightower. (#32, #872)
- "Vendorized" distlib as pip.vendor.distlib (https://distlib.readthedocs.io/).
- Fixed git VCS backend with git 1.8.3. (#967)

6.39 1.3.1 (2013-03-08)

- Fixed a major backward incompatible change of parsing URLs to externally hosted packages that got accidentally included in 1.3.

6.40 1.3 (2013-03-07)

- SSL Cert Verification; Make https the default for PyPI access. Thanks James Cleveland, Giovanni Bajo, Marcus Smith and many others. (#791, CVE-2013-1629)
- Added "pip list" for listing installed packages and the latest version available. Thanks Rafael Caricio, Miguel Araujo, Dmitry Gladkov. (#752)
- Fixed security issues with pip's use of temp build directories. Thanks David (d1b) and Thomas Guttler. (#780, CVE-2013-1888)
- Improvements to sphinx docs and cli help. (#773)
- Fixed an issue dealing with macOS temp dir handling, which was causing global NumPy installs to fail. (#707, #768)
- Split help output into general vs command-specific option groups. Thanks Georgi Valkov. (#744, #721)
- Fixed dependency resolution when installing from archives with uppercase project names. (#724)
- Fixed problem where re-installs always occurred when using file:// find-links. (#683, #702)
- "pip install -v" now shows the full download url, not just the archive name. Thanks Marc Abramowitz (#687)
- Fix to prevent unnecessary PyPI redirects. Thanks Alex Gronholm (#695)
- Fixed an install failure under Python 3 when the same version of a package is found under 2 different URLs. Thanks Paul Moore (#670, #671)
- Fix git submodule recursive updates. Thanks Roey Berman. (#674)
- Explicitly ignore rel='download' links while looking for html pages. Thanks Maxime R. (#677)
- –user/–upgrade install options now work together. Thanks 'eevee' for discovering the problem. (#705)
- Added check in `install --download` to prevent re-downloading if the target file already exists. Thanks Andrey Bulgakov. (#669)
- Added support for bare paths (including relative paths) as argument to *–find-links*. Thanks Paul Moore for draft patch.
- Added support for –no-index in requirements files.
- Added "pip show" command to get information about an installed package. Thanks Kelsey Hightower and Rafael Caricio. (#131)
- Added *–root* option for "pip install" to specify root directory. Behaves like the same option in distutils but also plays nice with pip's egg-info. Thanks Przemek Wrzos. (#253, #693)

6.41 1.2.1 (2012-09-06)

- Fixed a regression introduced in 1.2 about raising an exception when not finding any files to uninstall in the current environment. Thanks for the fix, Marcus Smith.

6.42 1.2 (2012-09-01)

- **Dropped support for Python 2.4** The minimum supported Python version is now Python 2.5.
- Fixed pypi mirror support being broken on some DNS responses. Thanks philwhin. (#605)

- Fixed pip uninstall removing files it didn't install. Thanks pjdelport. (#355)
- Fixed a number of issues related to improving support for the user installation scheme. Thanks Marcus Smith. (#493, #494, #440, #573)
- Write failure log to temp file if default location is not writable. Thanks andreigc.
- Pull in submodules for git editable checkouts. Thanks Hsiaoming Yang and Markus Hametner. (#289, #421)
- Use a temporary directory as the default build location outside of a virtualenv. Thanks Ben Rosser. (#339, #381)
- Added support for specifying extras with local editables. Thanks Nick Stenning.
- Added `--egg` flag to request egg-style rather than flat installation. Thanks Kamal Bin Mustafa. (#3)
- Prevent e.g. `gmpy2-2.0.tar.gz` from matching a request to `pip install gmpy`; sdist filename must begin with full project name followed by a dash. Thanks casevh for the report. (#510)
- Allow package URLS to have querystrings. Thanks W. Trevor King. (#504)
- pip freeze now falls back to non-editable format rather than blowing up if it can't determine the origin repository of an editable. Thanks Rory McCann. (#58)
- Added a *__main__.py* file to enable *python -m pip* on Python versions that support it. Thanks Alexey Luchko.
- Fixed upgrading from VCS url of project that does exist on index. Thanks Andrew Knapp for the report. (#487)
- Fix upgrade from VCS url of project with no distribution on index. Thanks Andrew Knapp for the report. (#486)
- Add a clearer error message on a malformed VCS url. Thanks Thomas Fenzl. (#427)
- Added support for using any of the built in guaranteed algorithms in `hashlib` as a checksum hash.
- Raise an exception if current working directory can't be found or accessed. (#321)
- Removed special casing of the user directory and use the Python default instead. (#82)
- Only warn about version conflicts if there is actually one. This re-enables using `==dev` in requirements files. (#436)
- Moved tests to be run on Travis CI: http://travis-ci.org/pypa/pip
- Added a better help formatter.

6.43 1.1 (2012-02-16)

- Don't crash when a package's setup.py emits UTF-8 and then fails. Thanks Marc Abramowitz. (#326)
- Added `--target` option for installing directly to arbitrary directory. Thanks Stavros Korokithakis.
- Added support for authentication with Subversion repositories. Thanks Qiangning Hong.
- `--download` now downloads dependencies as well. Thanks Qiangning Hong. (#315)
- Errors from subprocesses will display the current working directory. Thanks Antti Kaihola.
- Fixed compatibility with Subversion 1.7. Thanks Qiangning Hong. Note that setuptools remains incompatible with Subversion 1.7; to get the benefits of pip's support you must use Distribute rather than setuptools. (#369)
- Ignore py2app-generated macOS mpkg zip files in finder. Thanks Rene Dudfield. (#57)
- Log to ~/Library/Logs/ by default on macOS framework installs. Thanks Dan Callahan for report and patch. (#182)
- Understand version tags without minor version ("py3") in sdist filenames. Thanks Stuart Andrews for report and Olivier Girardot for patch. (#310)

- Pip now supports optionally installing setuptools "extras" dependencies; e.g. "pip install Paste[openid]". Thanks Matt Maker and Olivier Girardot. (#7)
- freeze no longer borks on requirements files with –index-url or –find-links. Thanks Herbert Pfennig. (#391)
- Handle symlinks properly. Thanks lebedov for the patch. (#288)
- pip install -U no longer reinstalls the same versions of packages. Thanks iguananaut for the pull request. (#49)
- Removed `-E`/`--environment` option and `PIP_RESPECT_VIRTUALENV`; both use a restart-in-venv mechanism that's broken, and neither one is useful since every virtualenv now has pip inside it. Replace `pip -E path/to/venv install Foo` with `virtualenv path/to/venv && path/to/venv/pip install Foo`.
- Fixed pip throwing an IndexError when it calls *scraped_rel_links*. (#366)
- pip search should set and return a useful shell status code. (#22)
- Added global `--exists-action` command line option to easier script file exists conflicts, e.g. from editable requirements from VCS that have a changed repo URL. (#351, #365)

6.44 1.0.2 (2011-07-16)

- Fixed docs issues.
- Reinstall a package when using the `install -I` option. (#295)
- Finds a Git tag pointing to same commit as origin/master. (#283)
- Use absolute path for path to docs in setup.py. (#279)
- Correctly handle exceptions on Python3. (#314)
- Correctly parse `--editable` lines in requirements files. (#320)

6.45 1.0.1 (2011-04-30)

- Start to use git-flow.
- *find_command* should not raise AttributeError. (#274)
- Respect Content-Disposition header. Thanks Bradley Ayers. (#273)
- pathext handling on Windows. (#233)
- svn+svn protocol. (#252)
- multiple CLI searches. (#44)
- Current working directory when running setup.py clean. (#266)

6.46 1.0 (2011-04-04)

- Added Python 3 support! Huge thanks to Vinay Sajip, Vitaly Babiy, Kelsey Hightower, and Alex Gronholm, among others.
- Download progress only shown on a real TTY. Thanks Alex Morega.
- Fixed finding of VCS binaries to not be fooled by same-named directories. Thanks Alex Morega.

- Fixed uninstall of packages from system Python for users of Debian/Ubuntu python-setuptools package (workaround until fixed in Debian and Ubuntu).
- Added get-pip.py installer. Simply download and execute it, using the Python interpreter of your choice:

```
$ curl -O https://raw.github.com/pypa/pip/master/contrib/get-pip.py
$ python get-pip.py
```

This may have to be run as root.

Note: Make sure you have distribute installed before using the installer!

6.47 0.8.3

- Moved main repository to Github: https://github.com/pypa/pip
- Transferred primary maintenance from Ian to Jannis Leidel, Carl Meyer, Brian Rosner
- Fixed no uninstall-on-upgrade with URL package. Thanks Oliver Tonnhofer. (#14)
- Fixed egg name not properly resolving. Thanks Igor Sobreira. (#163)
- Fixed Non-alphabetical installation of requirements. Thanks Igor Sobreira. (#178)
- Fixed documentation mentions –index instead of –index-url. Thanks Kelsey Hightower (#199)
- rmtree undefined in mercurial.py. Thanks Kelsey Hightower. (#204)
- Fixed bug in Git vcs backend that would break during reinstallation.
- Fixed bug in Mercurial vcs backend related to pip freeze and branch/tag resolution.
- Fixed bug in version string parsing related to the suffix "-dev".

6.48 0.8.2

- Avoid redundant unpacking of bundles (from pwaller)
- Fixed checking out the correct tag/branch/commit when updating an editable Git requirement. (#32, #150, #161)
- Added ability to install version control requirements without making them editable, e.g.:

```
pip install git+https://github.com/pypa/pip/
```

(#49)

- Correctly locate build and source directory on macOS. (#175)
- Added `git+https://` scheme to Git VCS backend.

6.49 0.8.1

- Added global –user flag as shortcut for –install-option="–user". From Ronny Pfannschmidt.
- Added support for PyPI mirrors as defined in PEP 381, from Jannis Leidel.

- Fixed git revisions being ignored. Thanks John-Scott Atlakson. (#138)
- Fixed initial editable install of github package from a tag failing. Thanks John-Scott Atlakson. (#95)
- Fixed installing if a directory in cwd has the same name as the package you're installing. (#107)
- –install-option="–prefix=~/.local" ignored with -e. Thanks Ronny Pfannschmidt and Wil Tan. (#39)

6.50 0.8

- Track which `build/` directories pip creates, never remove directories it doesn't create. From Hugo Lopes Tavares.
- Pip now accepts file:// index URLs. Thanks Dave Abrahams.
- Various cleanup to make test-running more consistent and less fragile. Thanks Dave Abrahams.
- Real Windows support (with passing tests). Thanks Dave Abrahams.
- `pip-2.7` etc. scripts are created (Python-version specific scripts)
- `contrib/build-standalone` script creates a runnable `.zip` form of pip, from Jannis Leidel
- Editable git repos are updated when reinstalled
- Fix problem with `--editable` when multiple `.egg-info/` directories are found.
- A number of VCS-related fixes for `pip freeze`, from Hugo Lopes Tavares.
- Significant test framework changes, from Hugo Lopes Tavares.

6.51 0.7.2

- Set zip_safe=False to avoid problems some people are encountering where pip is installed as a zip file.

6.52 0.7.1

- Fixed opening of logfile with no directory name. Thanks Alexandre Conrad.
- Temporary files are consistently cleaned up, especially after installing bundles, also from Alex Conrad.
- Tests now require at least ScriptTest 1.0.3.

6.53 0.7

- Fixed uninstallation on Windows
- Added `pip search` command.
- Tab-complete names of installed distributions for `pip uninstall`.
- Support tab-completion when there is a global-option before the subcommand.
- Install header files in standard (scheme-default) location when installing outside a virtualenv. Install them to a slightly more consistent non-standard location inside a virtualenv (since the standard location is a non-writable symlink to the global location).

- pip now logs to a central location by default (instead of creating `pip-log.txt` all over the place) and constantly overwrites the file in question. On Unix and macOS this is `'$HOME/.pip/pip.log'` and on Windows it's `'%HOME%\\pip\\pip.log'`. You are still able to override this location with the `$PIP_LOG_FILE` environment variable. For a complete (appended) logfile use the separate `'--log'` command line option.
- Fixed an issue with Git that left an editable package as a checkout of a remote branch, even if the default behaviour would have been fine, too.
- Fixed installing from a Git tag with older versions of Git.
- Expand "~" in logfile and download cache paths.
- Speed up installing from Mercurial repositories by cloning without updating the working copy multiple times.
- Fixed installing directly from directories (e.g. `pip install path/to/dir/`).
- Fixed installing editable packages with `svn+ssh` URLs.
- Don't print unwanted debug information when running the freeze command.
- Create log file directory automatically. Thanks Alexandre Conrad.
- Make test suite easier to run successfully. Thanks Dave Abrahams.
- Fixed "pip install ." and "pip install .."; better error for directory without setup.py. Thanks Alexandre Conrad.
- Support Debian/Ubuntu "dist-packages" in zip command. Thanks duckx.
- Fix relative –src folder. Thanks Simon Cross.
- Handle missing VCS with an error message. Thanks Alexandre Conrad.
- Added –no-download option to install; pairs with –no-install to separate download and installation into two steps. Thanks Simon Cross.
- Fix uninstalling from requirements file containing -f, -i, or –extra-index-url.
- Leftover build directories are now removed. Thanks Alexandre Conrad.

6.54 0.6.3

- Fixed import error on Windows with regard to the backwards compatibility package

6.55 0.6.2

- Fixed uninstall when /tmp is on a different filesystem.
- Fixed uninstallation of distributions with namespace packages.

6.56 0.6.1

- Added support for the `https` and `http-static` schemes to the Mercurial and `ftp` scheme to the Bazaar backend.
- Fixed uninstallation of scripts installed with easy_install.
- Fixed an issue in the package finder that could result in an infinite loop while looking for links.

- Fixed issue with `pip bundle` and local files (which weren't being copied into the bundle), from Whit Morriss.

6.57 0.6

- Add `pip uninstall` and uninstall-before upgrade (from Carl Meyer).
- Extended configurability with config files and environment variables.
- Allow packages to be upgraded, e.g., `pip install Package==0.1` then `pip install Package==0.2`.
- Allow installing/upgrading to Package==dev (fix "Source version does not match target version" errors).
- Added command and option completion for bash and zsh.
- Extended integration with virtualenv by providing an option to automatically use an active virtualenv and an option to warn if no active virtualenv is found.
- Fixed a bug with pip install –download and editable packages, where directories were being set with 0000 permissions, now defaults to 755.
- Fixed uninstallation of easy_installed console_scripts.
- Fixed uninstallation on macOS Framework layout installs
- Fixed bug preventing uninstall of editables with source outside venv.
- Creates download cache directory if not existing.

6.58 0.5.1

- Fixed a couple little bugs, with git and with extensions.

6.59 0.5

- Added ability to override the default log file name (`pip-log.txt`) with the environmental variable `$PIP_LOG_FILE`.
- Made the freeze command print installed packages to stdout instead of writing them to a file. Use simple redirection (e.g. `pip freeze > stable-req.txt`) to get a file with requirements.
- Fixed problem with freezing editable packages from a Git repository.
- Added support for base URLs using `<base href='...'>` when parsing HTML pages.
- Fixed installing of non-editable packages from version control systems.
- Fixed issue with Bazaar's bzr+ssh scheme.
- Added –download-dir option to the install command to retrieve package archives. If given an editable package it will create an archive of it.
- Added ability to pass local file and directory paths to `--find-links`, e.g. `--find-links=file:///path/to/my/private/archive`
- Reduced the amount of console log messages when fetching a page to find a distribution was problematic. The full messages can be found in pip-log.txt.

- Added `--no-deps` option to install ignore package dependencies
- Added `--no-index` option to ignore the package index (PyPI) temporarily
- Fixed installing editable packages from Git branches.
- Fixes freezing of editable packages from Mercurial repositories.
- Fixed handling read-only attributes of build files, e.g. of Subversion and Bazaar on Windows.
- When downloading a file from a redirect, use the redirected location's extension to guess the compression (happens specifically when redirecting to a bitbucket.org tip.gz file).
- Editable freeze URLs now always use revision hash/id rather than tip or branch names which could move.
- Fixed comparison of repo URLs so incidental differences such as presence/absence of final slashes or quoted/unquoted special characters don't trigger "ignore/switch/wipe/backup" choice.
- Fixed handling of attempt to checkout editable install to a non-empty, non-repo directory.

6.60 0.4

- Make `-e` work better with local hg repositories
- Construct PyPI URLs the exact way easy_install constructs URLs (you might notice this if you use a custom index that is slash-sensitive).
- Improvements on Windows (from Ionel Maries Cristian).
- Fixed problem with not being able to install private git repositories.
- Make `pip zip` zip all its arguments, not just the first.
- Fix some filename issues on Windows.
- Allow the `-i` and `--extra-index-url` options in requirements files.
- Fix the way bundle components are unpacked and moved around, to make bundles work.
- Adds `-s` option to allow the access to the global site-packages if a virtualenv is to be created.
- Fixed support for Subversion 1.6.

6.61 0.3.1

- Improved virtualenv restart and various path/cleanup problems on win32.
- Fixed a regression with installing from svn repositories (when not using `-e`).
- Fixes when installing editable packages that put their source in a subdirectory (like `src/`).
- Improve `pip -h`

6.62 0.3

- Added support for editable packages created from Git, Mercurial and Bazaar repositories and ability to freeze them. Refactored support for version control systems.

- Do not use `sys.exit()` from inside the code, instead use a return. This will make it easier to invoke programmatically.
- Put the install record in `Package.egg-info/installed-files.txt` (previously they went in `site-packages/install-record-Package.txt`).
- Fix a problem with `pip freeze` not including `-e svn+` when an svn structure is peculiar.
- Allow `pip -E` to work with a virtualenv that uses a different version of Python than the parent environment.
- Fixed Win32 virtualenv (`-E`) option.
- Search the links passed in with `-f` for packages.
- Detect zip files, even when the file doesn't have a `.zip` extension and it is served with the wrong Content-Type.
- Installing editable from existing source now works, like `pip install -e some/path/` will install the package in `some/path/`. Most importantly, anything that package requires will also be installed by pip.
- Add a `--path` option to `pip un/zip`, so you can avoid zipping files that are outside of where you expect.
- Add `--simulate` option to `pip zip`.

6.63 0.2.1

- Fixed small problem that prevented using `pip.py` without actually installing pip.
- Fixed `--upgrade`, which would download and appear to install upgraded packages, but actually just reinstall the existing package.
- Fixed Windows problem with putting the install record in the right place, and generating the `pip` script with Setuptools.
- Download links that include embedded spaces or other unsafe characters (those characters get %-encoded).
- Fixed use of URLs in requirement files, and problems with some blank lines.
- Turn some tar file errors into warnings.

6.64 0.2

- Renamed to `pip`, and to install you now do `pip install PACKAGE`
- Added command `pip zip PACKAGE` and `pip unzip PACKAGE`. This is particularly intended for Google App Engine to manage libraries to stay under the 1000-file limit.
- Some fixes to bundles, especially editable packages and when creating a bundle using unnamed packages (like just an svn repository without `#egg=Package`).

6.65 0.1.4

- Added an option `--install-option` to pass options to pass arguments to `setup.py install`
- `.svn/` directories are no longer included in bundles, as these directories are specific to a version of svn – if you build a bundle on a system with svn 1.5, you can't use the checkout on a system with svn 1.4. Instead a file `svn-checkout.txt` is included that notes the original location and revision, and the command you can use to turn it back into an svn checkout. (Probably unpacking the bundle should, maybe optionally, recreate this information – but that is not currently implemented, and it would require network access.)

- Avoid ambiguities over project name case, where for instance MyPackage and mypackage would be considered different packages. This in particular caused problems on Macs, where `MyPackage/` and `mypackage/` are the same directory.
- Added support for an environmental variable `$PIP_DOWNLOAD_CACHE` which will cache package downloads, so future installations won't require large downloads. Network access is still required, but just some downloads will be avoided when using this.

6.66 0.1.3

- Always use `svn checkout` (not `export`) so that `tag_svn_revision` settings give the revision of the package.
- Don't update checkouts that came from `.pybundle` files.

6.67 0.1.2

- Improve error text when there are errors fetching HTML pages when seeking packages.
- Improve bundles: include empty directories, make them work with editable packages.
- If you use `-E env` and the environment `env/` doesn't exist, a new virtual environment will be created.
- Fix `dependency_links` for finding packages.

6.68 0.1.1

- Fixed a NameError exception when running pip outside of a virtualenv environment.
- Added HTTP proxy support (from Prabhu Ramachandran)
- Fixed use of `hashlib.md5` on python2.5+ (also from Prabhu Ramachandran)

6.69 0.1

- Initial release

Index

Symbols

C