pip Documentation

Release 7.0.dev0

The pip developers

December 22, 2014

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
```

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List what packages are outdated: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$ pip list --outdated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 Python & OS Support

pip works with CPython versions 2.6, 2.7, 3.2, 3.3, 3.4 and also pypy.

pip works on Unix/Linux, OS X, and Windows.

Note: Python 2.5 was supported through v1.3.1, and Python 2.4 was supported through v1.1.

2.2 pip included with Python

Python 2.7.9 and later (on the python2 series), and Python 3.4 and later include pip by default [1], so you may have pip already.

2.3 Install pip

To install pip, securely download get-pip.py. [2]

Then run the following (which may require administrator access):

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

If setuptools (or distribute) is not already installed, `get-pip.py` will install setuptools for you. [3]

To upgrade an existing setuptools (or distribute), run `pip install -U setuptools`. [4]

Additionally, `get-pip.py` supports using the *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
```

[1] https://docs.python.org/3/installing/

[2] "Secure" in this context means using a modern browser or a tool like *curl* that verifies SSL certificates when downloading from https URLs.

[3] Beginning with pip v1.5.1, `get-pip.py` stopped requiring setuptools to be installed first.

[4] Although using `pip install --upgrade setuptools` to upgrade from distribute to setuptools works in isolation, it's possible to get "ImportError: No module named setuptools" when using pip<1.4 to upgrade a package that depends on setuptools or distribute. See `here for details`.

Install to the user site [5]:

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

Install behind a proxy:

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

2.4 Upgrade pip

On Linux or OS X:

```
pip install -U pip
```

On Windows [6]:

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

2.5 Using OS Package Managers

On Linux, pip will generally be available for the system install of python using the system package manager, although often the latest version will be unavailable.

On Debian and Ubuntu:

```
sudo apt-get install python-pip
```

On Fedora:

```
sudo yum install python-pip
```

---

[5] 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.

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

CHAPTER 3

User Guide

Contents

3.1 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.

3.2 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.

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)

3.3 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-use-wheel* 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
```

3.4 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.

3.5 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.

3.6 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.

3.7 Configuration

3.7.1 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: `$HOME/.config/pip/pip.conf` which respects the `XDG_CONFIG_HOME` environment variable.
- On Mac OS X the configuration file is `$HOME/Library/Application Support/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 Mac OS X 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 Mac OS X 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 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 Mac OS X the file is: `/Library/Application Support/pip/pip.conf`
- On Windows XP the file is: `C:\Documents and Settings\All Users\Application Data\PyPA\pip\pip.conf`
- On Windows 7 and later the file is hidden, but writeable at `C:\ProgramData\PyPA\pip\pip.conf`
- 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
```

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
```

3.7.2 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 [...]
```

To set options that can be set multiple times on the command line, just add spaces in between 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
```

3.7.3 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

3.7.4 Command Completion

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

To setup for bash:

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

To setup for zsh:

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

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

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

3.8 Fast & Local Installs

Often, you will want a fast install from local archives, without probing PyPI.

First, download the archives that fulfill your requirements:

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

Then, install using *–find-links* and *–no-index*:

```
$ pip install --no-index --find-links=[file://]<DIR> -r requirements.txt
```

3.9 “Only if needed” Recursive Upgrade

`pip install --upgrade` is currently written to perform an eager recursive upgrade, i.e. it upgrades all dependencies regardless of whether they still satisfy the new parent requirements.

E.g. supposing:

- *SomePackage-1.0* requires *AnotherPackage>=1.0*
- *SomePackage-2.0* requires *AnotherPackage>=1.0* and *OneMorePoject==1.0*
- *SomePackage-1.0* and *AnotherPackage-1.0* are currently installed
- *SomePackage-2.0* and *AnotherPackage-2.0* are the latest versions available on PyPI.

Running `pip install --upgrade SomePackage` would upgrade *SomePackage and AnotherPackage* despite *AnotherPackage* already being satisifed.

pip doesn't currently have an option to do an "only if needed" recursive upgrade, but you can achieve it using these 2 steps:

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

The first line will upgrade *SomePackage*, but not dependencies like *AnotherPackage*. The 2nd line will fill in new dependencies like *OneMorePackage*.

See #59 for a plan of making "only if needed" recursive the default behavior for a new `pip upgrade` command.

3.10 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
```

3.11 Ensuring Repeatability

Three things are required to fully guarantee a repeatable installation using requirements files.

1. The requirements file was generated by `pip freeze` or you're sure it only contains requirements that specify a specific version.
2. The installation is performed using *–no-deps*. This guarantees that only what is explicitly listed in the requirements file is installed.
3. The installation is performed against an index or find-links location that is guaranteed to *not* allow archives to be changed and updated without a version increase. Unfortunately, this is *not* true on PyPI. It is possible for the same pypi distribution to have a different hash over time. Project authors are allowed to delete a distribution, and then upload a new one with the same name and version, but a different hash. See Issue #1175 for plans to add hash confirmation to pip, or a new "lock file" notion, but for now, know that the peep project offers this feature on top of pip using requirements file comments.

3.12 Create an Installation Bundle with Compiled Dependencies

You can create a simple bundle that contains all of the dependencies you wish to install using:

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

Once you have a bundle, you can then uninstall it using:

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

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.

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.

Additionally, pip writes a "debug log" for every command. This log appends and will periodically rotate and clean itself up to limit on disk file size. The default location is as follows:

- On Unix: `$HOME/.cache/pip/log/debug.log`
- On Mac OS X: `$HOME/Library/Logs/pip/debug.log`
- On Windows: `C:\Users<username>AppDataLocalpipLogsdebug.log`

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`.

Build System Interface

Pip builds packages by invoking the build system. Presently, the only supported build system is `setuptools`, but future developments to the Python packaging infrastructure are expected to include support for other build systems. As well as package building, the build system is also invoked to install packages direct from source.

The 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.

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.

--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.

--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.

4.2 pip install

Contents

4.2.1 Usage

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

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:

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

Since version 6.0, pip also supports markers using the "; " separator. Examples:

```
futures; python_version < '2.7'
http://my.package.repo/SomePackage-1.0.4.zip; python_version >= '3.4'
```

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.

Additionally, the following Package Index Options are supported:

- *-i, –index-url*
- *–extra-index-url*
- *–no-index*
- *-f, –find-links*
- *–allow-external*
- *–allow-all-external*
- *–allow-unverified*
- *–no-use-wheel*

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
```

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

```
-r more_requirements.txt
```

Requirement Specifiers

pip supports installing from "requirement specifiers" as implemented in pkg_resources Requirements

Some Examples:

```
'FooProject >= 1.2'
Fizzy [foo, bar]
'PickyThing<1.6,>1.9,!=1.9.6,<2.0a0,==2.4c1'
SomethingWhoseVersionIDontCareAbout
```

Note: Use single or double quotes around specifiers when using them in a shell to avoid > and < being interpreted as shell redirects. e.g. `pip install 'FooProject>=1.2'`. Don't use single or double quotes in a `requirements.txt` file.

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.

Externally Hosted Files

Starting with v1.4, pip will warn about installing any file that does not come from the primary index. As of version 1.5, pip defaults to ignoring these files unless asked to consider them.

The `pip install` command supports a *–allow-external PROJECT* option that will enable installing links that are linked directly from the simple index but to an external host that also have a supported hash fragment. Externally hosted files for all projects may be enabled using the *–allow-all-external* flag to the `pip install` command.

The `pip install` command also supports a *–allow-unverified PROJECT* option that will enable installing insecurely linked files. These are either directly linked (as above) files without a hash, or files that are linked from either the home page or the download url of a package.

These options can be used in a requirements file. Assuming some fictional *ExternalPackage* that is hosted external and unverified, then your requirements file would be like so:

```
--allow-external ExternalPackage
--allow-unverified ExternalPackage
ExternalPackage
```

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.

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.

Git

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

Here are the supported forms:

```
[-e] git+git://git.myproject.org/MyProject#egg=MyProject
[-e] git+https://git.myproject.org/MyProject#egg=MyProject
[-e] git+ssh://git.myproject.org/MyProject#egg=MyProject
-e git+git@git.myproject.org:MyProject#egg=MyProject
```

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

```
[-e] git://git.myproject.org/MyProject.git@master#egg=MyProject
[-e] git://git.myproject.org/MyProject.git@v1.0#egg=MyProject
[-e] git://git.myproject.org/MyProject.git@da39a3ee5e6b4b0d3255bfef95601890afd80709#egg=MyProject
```

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
```

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

```
[-e] hg+http://hg.myproject.org/MyProject@da39a3ee5e6b#egg=MyProject
[-e] hg+http://hg.myproject.org/MyProject@2019#egg=MyProject
[-e] hg+http://hg.myproject.org/MyProject@v1.0#egg=MyProject
[-e] hg+http://hg.myproject.org/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.myproject.org/svn/MyProject#egg=MyProject
[-e] svn+http://svn.myproject.org/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.myproject.org/MyProject/trunk#egg=MyProject
[-e] bzr+sftp://user@myproject.org/MyProject/trunk#egg=MyProject
[-e] bzr+ssh://user@myproject.org/MyProject/trunk#egg=MyProject
[-e] bzr+ftp://user@myproject.org/MyProject/trunk#egg=MyProject
[-e] bzr+lp:MyProject#egg=MyProject
```

Tags or revisions can be installed like so:

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

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.

See the *pip install Examples*.

SSL Certificate Verification

Starting with v1.3, pip provides SSL certificate verification over https, for the purpose of providing secure, certified downloads from PyPI.

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 it's 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 all together. If you want a fast/local install solution that circumvents accessing PyPI, see *Fast & Local Installs*.

Hash Verification

PyPI provides md5 hashes in the hash fragment of package download urls.

pip supports checking this, as well as any of the guaranteed hashlib algorithms (sha1, sha224, sha384, sha256, sha512, md5).

The hash fragment is case sensitive (i.e. sha1 not SHA1).

This check is only intended to provide basic download corruption protection. It is not intended to provide security against tampering. For that, see *SSL Certificate Verification*

"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-comp
```

The `egg_info` command should create egg metadata for the package, as described in the setuptools documentation at http://pythonhosted.org/setuptools/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.

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

-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.

-b, --build <dir>
: Directory to unpack packages into and build in.

-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.

-d, --download <dir>
Download packages into <dir> instead of installing them, regardless of what's already installed.

--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. This process is recursive regardless of whether a dependency is already satisfied.

--force-reinstall
When upgrading, reinstall all packages even if they are already up-to-date.

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

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

--no-install
DEPRECATED. Download and unpack all packages, but don't actually install them.

--no-download
DEPRECATED. Don't download any packages, just install the ones already downloaded (completes an install run with –no-install).

--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.

--user
Install using the user scheme.

--egg
Install packages as eggs, not 'flat', like pip normally does. This option is not about installing *from* eggs. (WARNING: Because this option overrides pip's normal install logic, requirements files may not behave as expected.)

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

--compile
Compile py files to pyc

--no-compile
Do not compile py files to pyc

--no-use-wheel
Do not Find and prefer wheel archives when searching indexes and find-links locations.

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

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

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

--extra-index-url <url>
Extra URLs of package indexes to use in addition to –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.

--allow-external <package>
Allow the installation of a package even if it is externally hosted

--allow-all-external
Allow the installation of all packages that are externally hosted

--allow-unverified <package>
Allow the installation of a package even if it is hosted in an insecure and unverifiable way

--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' bra
$ pip install -e git+https://git.repo/some_repo.git#egg=subdir&subdirectory=subdir_path # instal
```

6. Install a package with setuptools extras.

```
$ pip install SomePackage[PDF]
$ pip install SomePackage[PDF]==3.0
$ pip install -e .[PDF]==3.0  # editable project in current directory
```

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 uninstall

Contents

4.3.1 Usage

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

4.3.2 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`.

4.3.3 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.

4.3.4 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
```

4.4 pip freeze

Contents

4.4.1 Usage

```
pip freeze [options]
```

4.4.2 Description

Output installed packages in requirements format.

packages are listed in a case-insensitive sorted order.

4.4.3 Options

-r, --requirement <file>
Use the order in the given requirements file and its comments when generating output.

-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.

4.4.4 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
```

4.5 pip list

Contents

4.5.1 Usage

```
pip list [options]
```

4.5.2 Description

List installed packages, including editables.

Packages are listed in a case-insensitive sorted order.

4.5.3 Options

-o, --outdated
List outdated packages (excluding editables)

-u, --uptodate
List uptodate packages (excluding editables)

-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.

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

--extra-index-url <url>
Extra URLs of package indexes to use in addition to –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.

--allow-external <package>
Allow the installation of a package even if it is externally hosted

--allow-all-external
Allow the installation of all packages that are externally hosted

--allow-unverified <package>
Allow the installation of a package even if it is hosted in an insecure and unverifiable way

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

4.5.4 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)
```

4.6 pip show

Contents

4.6.1 Usage

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

4.6.2 Description

Show information about one or more installed packages.

4.6.3 Options

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

4.6.4 Examples

1. Show information about a package:

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

4.7 pip search

Contents

4.7.1 Usage

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

4.7.2 Description

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

4.7.3 Options

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

4.7.4 Examples

1. Search for "peppercorn"

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

4.8 pip wheel

Contents

4.8.1 Usage

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

4.8.2 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: http://wheel.readthedocs.org/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.

4.8.3 Options

-w, --wheel-dir <dir>
: Build wheels into <dir>, where the default is '<cwd>/wheelhouse'.

--no-use-wheel
: Do not Find and prefer wheel archives when searching indexes and find-links locations.

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

-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".

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

-b, --build <dir>
Directory to unpack packages into and build in.

--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.

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

--extra-index-url <url>
Extra URLs of package indexes to use in addition to –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.

--allow-external <package>
Allow the installation of a package even if it is externally hosted

--allow-all-external
Allow the installation of all packages that are externally hosted

--allow-unverified <package>
Allow the installation of a package even if it is hosted in an insecure and unverifiable way

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

4.8.4 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
```

CHAPTER 5

Development

5.1 Pull Requests

- Submit Pull Requests against the *develop* 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 'develop' 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
```

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 Release Process

This process includes virtualenv, since pip releases necessitate a virtualenv release.

As an example, the instructions assume we're releasing pip-1.4, and virtualenv-1.10.

1. Upgrade setuptools, if needed:

1. Upgrade setuptools in `virtualenv/develop` using the *Refresh virtualenv* process.
2. Create a pull request against `pip/develop` with a modified `.travis.yml` file that installs virtualenv from `virtualenv/develop`, to confirm the travis builds are still passing.

2. Create Release branches:

1. Create `pip/release-1.4` branch.
2. In `pip/develop`, change `pip.version` to '1.5.dev1'.
3. Create `virtualenv/release-1.10` branch.
4. In `virtualenv/develop`, change `virtualenv.version` to '1.11.dev1'.

3. Prepare "rcX":

1. In `pip/release-1.4`, change `pip.version` to '1.4rcX', and tag with '1.4rcX'.
2. Build a pip sdist from `pip/release-1.4`, and build it into `virtualenv/release-1.10` using the *Refresh virtualenv* process.
3. In `virtualenv/release-1.10`, change `virtualenv.version` to '1.10rcX', and tag with '1.10rcX'.

4. Announce `pip-1.4rcX` and `virtualenv-1.10rcX` with the *RC Install Instructions* and elicit feedback.
5. Apply fixes to 'rcX':

1. Apply fixes to `pip/release-1.4` and `virtualenv/release-1.10`
2. Periodically merge fixes to `pip/develop` and `virtualenv/develop`

6. Repeat #4 thru #6 if needed.
7. Final Release:

1. In `pip/release-1.4`, change `pip.version` to '1.4', and tag with '1.4'.
2. Merge `pip/release-1.4` to `pip/master`.
3. Build a pip sdist from `pip/release-1.4`, and load it into `virtualenv/release-1.10` using the *Refresh virtualenv* process.
4. Merge `vitualenv/release-1.10` to `virtualenv/develop`.
5. In `virtualenv/release-1.10`, change `virtualenv.version` to '1.10', and tag with '1.10'.
6. Merge `virtualenv/release-1.10` to `virtualenv/master`
7. Build and upload pip and virtualenv sdists to PyPI.

5.5.1 Refresh virtualenv

1. Update the embedded versions of pip and setuptools in `virtualenv_support`.
2. Run `bin/rebuild-script.py` to rebuild virtualenv based on the latest versions.

5.5.2 RC Install Instructions

```
$ curl -L -O https://github.com/pypa/virtualenv/archive/1.10rc1.tar.gz
$ echo "<md5sum value>  1.10rc1.tar.gz" | md5sum -c
1.10rc1.tar.gz: OK
$ tar zxf 1.10rc1.tar.gz
$ python virtualenv-1.10rc1/virtualenv.py myVE
$ myVE/bin/pip install SomePackage
```

CHAPTER 6

Release Notes

7.0.dev1 (unreleased)

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.

6.0 (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. (PR #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. (PR #1933)
- **BACKWARD INCOMPATIBLE** The ability to install files over the FTP protocol was accidently lost in pip 1.5 and it has now been decided to not restore that ability.
- **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/latest/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. See #906 for discussion.
- **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.
- Fixed #1873. Silence byte compile errors when installation succeed.
- Added a virtualenv-specific configuration file. (PR #1364)

- Added site-wide configuation files. (PR #1978)
- Added an automatic check to warn if there is an updated version of pip available (PR #2049).
- *wsgiref* and *argparse* (for >py26) are now excluded from *pip list* and *pip freeze* (PR #1606, PR #1369)
- Fixed #1424. Add `--client-cert` option for SSL client certificates.
- Fixed #1484. *pip show –files* was broken for wheel installs. (PR #1635)
- Fixed #1641. install_lib should take precedence when reading distutils config. (PR #1642)
- Send *Accept-Encoding: identity* when downloading files in an attempt to convince some servers who double compress the downloaded file to stop doing so. (PR #1688)
- Fixed #1559. Stop breaking when given pip commands in uppercase (PR #1725)
- Fixed #1618. Pip no longer adds duplicate logging consumers, so it won't create duplicate output when being called multiple times. (PR #1723)
- Fixed #1769. *pip wheel* now returns an error code if any wheels fail to build.
- Fixed #1775. *pip wheel* wasn't building wheels for dependencies of editable requirements.
- Allow the use of `--no-use-wheel` within a requirements file. (PR #1859)
- Fixed #1680. Attempt to locate system TLS certificates to use instead of the included CA Bundle if possible. (PR #1866)
- Fixed #1319. Allow use of Zip64 extension in Wheels and other zip files. (PR #1868)
- Fixed #1101. Properly handle an index or –find-links target which has a <base> without a href attribute. (PR #1869)
- Fixed #1885. Properly handle extras when a project is installed via Wheel. (PR #1896)
- Fixed #1180. Added support to respect proxies in `pip search`. It also fixes #932 and #1104. (PR #1902)
- Fixed #798 and #1060. *pip install –download* works with vcs links. (PR #1926)
- Fixed #1456. Disabled warning about insecure index host when using localhost. Based off of Guy Rozendorn's work in PR #1718. (PR #1967)
- Allow the use of OS standard user configuration files instead of ones simply based around `$HOME`. (PR #2021)
- Fixed #1825. When installing directly from wheel paths or urls, previous versions were not uninstalled. This also fixes #804 specifically for the case of wheel archives. (PR #1838)
- Fixed #2075, 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. (PR #2076)
- Fixed #1964, #1935, #676, Use a randomized and secure default build directory when possible. (PR #2122, CVE-2014-8991)
- Fixed #1433. Support environment markers in requirements.txt files. (pull:*2134*)
- Automatically retry failed HTTP requests by default. (PR #1444, pull:*2147*)
- Fixed #1100 - Handle HTML Encoding better using a method that is more similar to how browsers handle it. (PR #1874)
- Reduce the verbosity of the pip command by default. (PR #2175, PR #2177, PR #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 (PR #2183).

- Support setuptools style environment markers in a source distribution (PR #2153).

1.5.6 (2014-05-16)

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

1.5.5 (2014-05-03)

- Fixes #1632. Uninstall issues on debianized pypy, specifically issues with setuptools upgrades. (PR #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).

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.

1.5.3 (2014-02-20)

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

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

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

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`. (PR #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`. (PR #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. (PR #1251)
- Pip now confirms a wheel is supported when installing directly from a path or url. (PR #1315)
- Fixed #1097, `--ignore-installed` now behaves again as designed, after it was unintentionally broke in v0.8.3 when fixing #14 (PR #1352).
- Fixed a bug where global scripts were being removed when uninstalling –user installed packages (PR #1353).
- Fixed #1163, –user wasn't being respected when installing scripts from wheels (PR #1176).
- Fixed #1150, we now assume '_' means '-' in versions from wheel filenames (PR #1158).
- Fixed #219, error when using –log with a failed install (PR #1205).
- Fixed #1131, logging was buffered and choppy in Python 3.
- Fixed #70, –timeout was being ignored (PR #1202).
- Fixed #772, error when setting PIP_EXISTS_ACTION (PR #1201).
- Added colors to the logging output in order to draw attention to important warnings and errors. (PR #1109)
- Added warnings when using an insecure index, find-link, or dependency link. (PR #1121)
- Added support for installing packages from a subdirectory using the `subdirectory` editable option. (PR #1082)
- Fixed #1192. "TypeError: bad operand type for unary" in some cases when installing wheels using –find-links (PR #1218).
- Fixed #1133 and #317. 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 (PR #1146).
- PreviousBuildDirError now returns a non-zero exit code and prevents the previous build dir from being cleaned in all cases (PR #1162).
- Renamed –allow-insecure to –allow-unverified, however the old name will continue to work for a period of time (PR #1257).
- Fixed #1006, error when installing local projects with symlinks in Python 3. (PR #1311)
- The previously hidden `--log-file` otion, is now shown as a general option. (PR #1316)

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 (PR #1116).
- Fixed error when sysconfig module throws an exception (PR #1095).

- Don't ignore already installed pre-releases (PR #1076).
- Fixes related to upgrading setuptools (PR #1092).
- Fixes so that –download works with wheel archives (PR #1113).
- Fixes related to recognizing and cleaning global build dirs (PR #1080).

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. (PR #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 (PR #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'. (PR #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. (PR #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.
- Fixed a number of issues (#413, #709, #634, #602, and #939) related to cleaning up and not reusing build directories. (PR #865, #948)
- Added a User Agent so that pip is identifiable in logs. (PR #901)
- Added ssl and –user support to get-pip.py. Thanks Gabriel de Perthuis. (PR #895)
- Fixed the proxy support, which was broken in pip 1.3.x (PR #840)
- Fixed #32 - pip fails when server does not send content-type header. Thanks Hugo Lopes Tavares and Kelsey Hightower (PR #872).
- "Vendorized" distlib as pip.vendor.distlib (https://distlib.readthedocs.org/).
- Fixed git VCS backend with git 1.8.3. (PR #967)

1.3.1 (2013-03-08)

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

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 (PR #791, CVE-2013-1629).
- Added "pip list" for listing installed packages and the latest version available. Thanks Rafael Caricio, Miguel Araujo, Dmitry Gladkov (PR #752)
- Fixed security issues with pip's use of temp build directories. Thanks David (d1b) and Thomas Guttler. (PR #780, CVE-2013-1888)
- Improvements to sphinx docs and cli help. (PR #773)

- Fixed #707, dealing with OS X temp dir handling, which was causing global NumPy installs to fail. (PR #768)
- Split help output into general vs command-specific option groups. Thanks Georgi Valkov. (PR #744; PR #721 contains preceding refactor)
- Fixed dependency resolution when installing from archives with uppercase project names. (PR #724)
- Fixed problem where re-installs always occurred when using file:// find-links. (Pulls #683/#702)
- "pip install -v" now shows the full download url, not just the archive name. Thanks Marc Abramowitz (PR #687)
- Fix to prevent unnecessary PyPI redirects. Thanks Alex Gronholm (PR #695)
- Fixed #670 - install failure under Python 3 when the same version of a package is found under 2 different URLs. Thanks Paul Moore (PR #671)
- Fix git submodule recursive updates. Thanks Roey Berman. (Pulls #674)
- Explicitly ignore rel='download' links while looking for html pages. Thanks Maxime R. (PR #677)
- –user/–upgrade install options now work together. Thanks 'eevee' for discovering the problem. (PR #705)
- Added check in `install --download` to prevent re-downloading if the target file already exists. Thanks Andrey Bulgakov. (PR #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. Fixes #131. Thanks Kelsey Hightower and Rafael Caricio.
- 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 / PR #693)

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.

1.2 (2012-09-01)

- **Dropped support for Python 2.4** The minimum supported Python version is now Python 2.5.
- Fixed #605 - pypi mirror support broken on some DNS responses. Thanks philwhin.
- Fixed #355 - pip uninstall removes files it didn't install. Thanks pjdelport.
- Fixed issues #493, #494, #440, and #573 related to improving support for the user installation scheme. Thanks Marcus Smith.
- Write failure log to temp file if default location is not writable. Thanks andreigc.
- Pull in submodules for git editable checkouts. Fixes #289 and #421. Thanks Hsiaoming Yang and Markus Hametner.
- Use a temporary directory as the default build location outside of a virtualenv. Fixes issues #339 and #381. Thanks Ben Rosser.
- Added support for specifying extras with local editables. Thanks Nick Stenning.
- Added `--egg` flag to request egg-style rather than flat installation. Refs #3. Thanks Kamal Bin Mustafa.
- Fixed #510 - 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.

- Fixed #504 - allow package URLS to have querystrings. Thanks W. Trevor King.
- Fixed #58 - 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.
- Added a *__main__.py* file to enable *python -m pip* on Python versions that support it. Thanks Alexey Luchko.
- Fixed #487 - upgrade from VCS url of project that does exist on index. Thanks Andrew Knapp for the report.
- Fixed #486 - fix upgrade from VCS url of project with no distribution on index. Thanks Andrew Knapp for the report.
- Fixed #427 - clearer error message on a malformed VCS url. Thanks Thomas Fenzl.
- Added support for using any of the built in guaranteed algorithms in `hashlib` as a checksum hash.
- Fixed #321 - Raise an exception if current working directory can't be found or accessed.
- Fixed #82 - Removed special casing of the user directory and use the Python default instead.
- Fixed #436 - Only warn about version conflicts if there is actually one. This re-enables using `==dev` in requirements files.
- Moved tests to be run on Travis CI: http://travis-ci.org/pypa/pip
- Added a better help formatter.

1.1 (2012-02-16)

- Fixed #326 - don't crash when a package's setup.py emits UTF-8 and then fails. Thanks Marc Abramowitz.
- Added `--target` option for installing directly to arbitrary directory. Thanks Stavros Korokithakis.
- Added support for authentication with Subversion repositories. Thanks Qiangning Hong.
- Fixed #315 - `--download` now downloads dependencies as well. Thanks Qiangning Hong.
- Errors from subprocesses will display the current working directory. Thanks Antti Kaihola.
- Fixed #369 - 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.
- Fixed #57 - ignore py2app-generated OS X mpkg zip files in finder. Thanks Rene Dudfield.
- Fixed #182 - log to ~/Library/Logs/ by default on OS X framework installs. Thanks Dan Callahan for report and patch.
- Fixed #310 - understand version tags without minor version ("py3") in sdist filenames. Thanks Stuart Andrews for report and Olivier Girardot for patch.
- Fixed #7 - Pip now supports optionally installing setuptools "extras" dependencies; e.g. "pip install Paste[openid]". Thanks Matt Maker and Olivier Girardot.
- Fixed #391 - freeze no longer borks on requirements files with –index-url or –find-links. Thanks Herbert Pfennig.
- Fixed #288 - handle symlinks properly. Thanks lebedov for the patch.
- Fixed #49 - pip install -U no longer reinstalls the same versions of packages. Thanks iguananaut for the pull request.
- 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 #366 - pip throws IndexError when it calls *scraped_rel_links*

- Fixed #22 - pip search should set and return a userful shell status code
- Fixed #351 and #365 - 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.

1.0.2 (2011-07-16)

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

1.0.1 (2011-04-30)

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

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!

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 #14 - No uninstall-on-upgrade with URL package. Thanks Oliver Tonnhofer
- Fixed #163 - Egg name not properly resolved. Thanks Igor Sobreira
- Fixed #178 - Non-alphabetical installation of requirements. Thanks Igor Sobreira

- Fixed #199 - Documentation mentions –index instead of –index-url. Thanks Kelsey Hightower
- Fixed #204 - rmtree undefined in mercurial.py. Thanks Kelsey Hightower
- 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".

0.8.2

- Avoid redundant unpacking of bundles (from pwaller)
- Fixed #32, #150, #161 - Fixed checking out the correct tag/branch/commit when updating an editable Git requirement.
- Fixed #49 - Added ability to install version control requirements without making them editable, e.g.:

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

- Fixed #175 - Correctly locate build and source directory on Mac OS X.
- Added `git+https://` scheme to Git VCS backend.

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 #138 - Git revisions ignored. Thanks John-Scott Atlakson.
- Fixed #95 - Initial editable install of github package from a tag fails. Thanks John-Scott Atlakson.
- Fixed #107 - Can't install if a directory in cwd has the same name as the package you're installing.
- Fixed #39 - –install-option="–prefix=~/.local" ignored with -e. Thanks Ronny Pfannschmidt and Wil Tan.

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.

0.7.2

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

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.

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 Mac OS X 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 packge 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.

0.6.3

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

0.6.2

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

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.

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 Mac OS X Framework layout installs
- Fixed bug preventing uninstall of editables with source outside venv.
- Creates download cache directory if not existing.

0.5.1

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

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.

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.

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`

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`.

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.

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`).

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.

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.

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.

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)

0.1

- Initial release

Symbols

C