pip-accel

Release 0.43

Sep 27, 2017

Contents

The pip accelerator makes pip (the Python package manager) faster by keeping pip off the internet when possible and by caching compiled binary distributions. It can bring a 10 minute run of `pip` down to less than a minute. You can find the pip accelerator in the following places:

- The source code lives on GitHub
- Downloads are available in the Python Package Index
- Online documentation is hosted by Read The Docs

This is the documentation for version 0.43 of the pip accelerator. The documentation consists of two parts:

- The documentation for users of the `pip-accel` command
- The documentation for developers who wish to extend and/or embed the functionality of `pip-accel`

CHAPTER 1

Introduction & usage

The first part of the documentation is the readme which is targeted at users of the `pip-accel` command. Here are the topics discussed in the readme:

pip-accel: Accelerator for pip, the Python package manager

The pip-accel program is a wrapper for pip, the Python package manager. It accelerates the usage of pip to initialize Python virtual environments given one or more requirements files. It does so by combining the following two approaches:

1. Source distribution downloads are cached and used to generate a local index of source distribution archives. If all your dependencies are pinned to absolute versions whose source distribution downloads were previously cached, pip-accel won't need a network connection at all! This is one of the reasons why pip can be so slow: given absolute pinned dependencies available in the download cache it will still scan PyPI and distribution websites.
2. Binary distributions are used to speed up the process of installing dependencies with binary components (like M2Crypto and LXML). Instead of recompiling these dependencies again for every virtual environment we compile them once and cache the result as a binary `*.tar.gz` distribution.

In addition, since version 0.9 pip-accel contains a simple mechanism that detects missing system packages when a build fails and prompts the user whether to install the missing dependencies and retry the build.

The pip-accel program is currently tested on cPython 2.6, 2.7, 3.4 and 3.5 and PyPy (2.7). The automated test suite regularly runs on Ubuntu Linux (Travis CI) as well as Microsoft Windows (AppVeyor). In addition to these platforms pip-accel should work fine on most UNIX systems (e.g. Mac OS X).

Contents

Status

Paylogic uses pip-accel to quickly and reliably initialize virtual environments on its farm of continuous integration slaves which are constantly running unit tests (this was one of the original use cases for which pip-accel was developed). We also use it on our build servers.

When pip-accel was originally developed PyPI was sometimes very unreliable (PyPI wasn't behind a CDN back then). Because of the CDN, PyPI is much more reliable nowadays however pip-accel still has its place:

- The CDN doesn't help for distribution sites, which are as unreliably as they have always been.
- By using pip-accel you can make Python deployments completely independent from internet connectivity.
- Because pip-accel caches compiled binary packages it can still provide a nice speed boost over using plain pip.

Usage

The pip-accel command supports all subcommands and options supported by pip, however it is of course only useful for the `pip install` subcommand. So for example:

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

Alternatively you can also run pip-accel as follows, but note that this requires Python 2.7 or higher (it specifically doesn't work on Python 2.6):

```
$ python -m pip_accel install -r requirements.txt
```

If you pass a `-v` or `--verbose` option then pip and pip-accel will both use verbose output. The `-q` or `--quiet` option is also supported.

Based on the user running pip-accel the following file locations are used by default:

Root user	All other users	Purpose
`/var/cache/pip-accel`	`~/.pip-accel`	Used to store the source/binary indexes

This default can be overridden by defining the environment variable `PIP_ACCEL_CACHE`.

Configuration

For most users the default configuration of pip-accel should be fine. If you do want to change pip-accel's defaults you do so by setting environment variables and/or adding configuration options to a configuration file. This is because pip-accel shares its command line interface with pip and adding support for command line options specific to pip-accel is non trivial and may end up causing more confusion than it's worth :-). For an overview of the available configuration options and corresponding environment variables please refer to the documentation of the pip_accel.config module.

How fast is it?

To give you an idea of how effective pip-accel is, below are the results of a test to build a virtual environment for one of the internal code bases of Paylogic. This code base requires more than 40 dependencies including several packages that need compilation with SWIG and a C compiler:

Program	Description	Duration	Percentage
pip	Default configuration	444 seconds	100% (baseline)
pip	With download cache (first run)	416 seconds	94%
pip	With download cache (second run)	318 seconds	72%
pip-accel	First run	397 seconds	89%
pip-accel	Second run	30 seconds	7%

Alternative cache backends

Bundled with pip-accel are a local cache backend (which stores distribution archives on the local file system) and an Amazon S3 backend (see below).

Both of these cache backends are registered with pip-accel using a generic pluggable cache backend registration mechanism. This mechanism makes it possible to register additional cache backends without modifying pip-accel. If you are interested in the details please refer to pip-accel's `setup.py` script and the two simple Python modules that define the bundled backends.

If you've written a cache backend that you think may be valuable to others, please feel free to open an issue or pull request on GitHub in order to get your backend bundled with pip-accel.

Storing the binary cache on Amazon S3

You can configure pip-accel to store its binary cache files in an Amazon S3 bucket. In this case Amazon S3 is treated as a second level cache, only used if the local file system cache can't satisfy a dependency. If the dependency is not found in the Amazon S3 bucket, the package is built and cached locally (as usual) but then also saved to the Amazon S3 bucket. This functionality can be useful for continuous integration build worker boxes that are ephemeral and don't have persistent local storage to store the pip-accel binary cache.

To get started you need to install pip-accel as follows:

```
$ pip install 'pip-accel[s3]'
```

The `[s3]` part enables the Amazon S3 cache backend by installing the Boto package. Once installed you can use the following environment variables to configure the Amazon S3 cache backend:

`$PIP_ACCEL_S3_BUCKET` The name of the Amazon S3 bucket in which binary distribution archives should be cached. This environment variable is required to enable the Amazon S3 cache backend.

`$PIP_ACCEL_S3_PREFIX` The optional prefix to apply to all Amazon S3 keys. This enables name spacing based on the environment in which pip-accel is running (to isolate the binary caches of ABI incompatible systems). *The user is currently responsible for choosing a suitable prefix.*

$PIP_ACCEL_S3_READONLY If this option is set pip-accel will skip uploading to the Amazon S3 bucket. This means pip-accel will use the configured Amazon S3 bucket to "warm up" your local cache but it will never write to the bucket, so you can use read only credentials. Of course you will need to run at least one instance of pip-accel that does have write permissions, so this setup is best suited to teams working around e.g. a continuous integration (CI) server, where the CI server primes the cache and developers use the cache in read only mode.

You can also set these options from a configuration file, please refer to the documentation of the pip_accel.config module. You will also need to set AWS credentials, either in a .boto file or in the `$AWS_ACCESS_KEY_ID` and `$AWS_SECRET_ACCESS_KEY` environment variables (refer to the Boto documentation for details).

Using S3 compatible storage services

If you want to point pip-accel at an S3 compatible storage service that is *not* Amazon S3 you can override the S3 API URL using a configuration option or environment variable. For example the pip-accel test suite first installs and starts FakeS3 and then sets `PIP_ACCEL_S3_URL=http://localhost:12345` to point pip-accel at the FakeS3 server (in order to test the Amazon S3 cache backend without actually having to pay for an Amazon S3 bucket :-). For more details please refer to the documentation of the Amazon S3 cache backend.

Caching of setup requirements

Since version 0.38 pip-accel instructs setuptools to cache setup requirements in a subdirectory of pip-accel's data directory (see the eggs_cache option) to avoid recompilation of setup requirements. This works by injecting a symbolic link called `.eggs` into unpacked source distribution directories before pip or pip-accel runs the setup script.

The use of the `.eggs` directory was added in setuptools version 7.0 which is why pip-accel now requires setuptools 7.0 or higher to be installed. This dependency was added because the whole point of pip-accel is to work well out of the box, shielding the user from surprising behavior like setup requirements slowing things down and breaking offline installation.

Dependencies on system packages

Since version 0.9 pip-accel contains a simple mechanism that detects missing system packages when a build fails and prompts the user whether to install the missing dependencies and retry the build. Currently only Debian Linux and derivative Linux distributions are supported, although support for other platforms should be easy to add. This functionality currently works based on configuration files that define dependencies of Python packages on system packages. This means the results should be fairly reliable, but every single dependency needs to be manually defined...

Here's what it looks like in practice:

```
2013-06-16 01:01:53 wheezy-vm INFO Building binary distribution of python-mcrypt (1.
↪1) ..
2013-06-16 01:01:53 wheezy-vm ERROR Failed to build binary distribution of python-
↪mcrypt! (version: 1.1)
2013-06-16 01:01:53 wheezy-vm INFO Build output (will probably provide a hint as to␣
↪what went wrong):

gcc -pthread -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -
↪fPIC -DVERSION="1.1" -I/usr/include/python2.7 -c mcrypt.c -o build/temp.linux-i686-
↪2.7/mcrypt.o
mcrypt.c:23:20: fatal error: mcrypt.h: No such file or directory
error: command 'gcc' failed with exit status 1

2013-06-16 01:01:53 wheezy-vm INFO python-mcrypt: Checking for missing dependencies ..
2013-06-16 01:01:53 wheezy-vm INFO You seem to be missing 1 dependency: libmcrypt-dev
```

```
2013-06-16 01:01:53 wheezy-vm INFO I can install it for you with this command: sudo␣
↪apt-get install --yes libmcrypt-dev
Do you want me to install this dependency? [y/N] y
2013-06-16 01:02:05 wheezy-vm INFO Got permission to install missing dependency.

The following extra packages will be installed:
  libmcrypt4
Suggested packages:
  mcrypt
The following NEW packages will be installed:
  libmcrypt-dev libmcrypt4
0 upgraded, 2 newly installed, 0 to remove and 68 not upgraded.
Unpacking libmcrypt4 (from .../libmcrypt4_2.5.8-3.1_i386.deb) ...
Unpacking libmcrypt-dev (from .../libmcrypt-dev_2.5.8-3.1_i386.deb) ...
Setting up libmcrypt4 (2.5.8-3.1) ...
Setting up libmcrypt-dev (2.5.8-3.1) ...

2013-06-16 01:02:13 wheezy-vm INFO Successfully installed 1 missing dependency.
2013-06-16 01:02:13 wheezy-vm INFO Building binary distribution of python-mcrypt (1.
↪1) ..
2013-06-16 01:02:14 wheezy-vm INFO Copying binary distribution python-mcrypt-1.1.
↪linux-i686.tar.gz to cache as python-mcrypt:1.1:py2.7.tar.gz.
```

Integrating with tox

You can tell Tox to use pip-accel using a small shell script that first uses pip to install pip-accel, then uses pip-accel to bootstrap the virtual environment. You can find details about this in issue #30 on GitHub.

Control flow of pip-accel

The way pip-accel works is not very intuitive but it is very effective. Below is an overview of the control flow. Once you take a look at the code you'll notice that the steps below are all embedded in a loop that retries several times. This is mostly because of step 2 (downloading the source distributions).

1. Run `pip install --download=... --no-index -r requirements.txt` to unpack source distributions available in the local source index. This is the first step because pip-accel should accept *requirements.txt* files as input but it will manually install dependencies from cached binary distributions (without using pip or easy_install):

   - If the command succeeds it means all dependencies are already available as downloaded source distributions. We'll parse the verbose pip output of step 1 to find the direct and transitive dependencies (names and versions) defined in *requirements.txt* and use them as input for step 3. Go to step 3.
   - If the command fails it probably means not all dependencies are available as local source distributions yet so we should download them. Go to step 2.

2. Run `pip install --download=... -r requirements.txt` to download missing source distributions to the download cache:

   - If the command fails it means that pip encountered errors while scanning PyPI, scanning a distribution website, downloading a source distribution or unpacking a source distribution. Usually these kinds of errors are intermittent so retrying a few times is worth a shot. Go to step 2.
   - If the command succeeds it means all dependencies are now available as local source distributions; we don't need the network anymore! Go to step 1.

3. Run `python setup.py bdist_dumb --format=gztar` for each dependency that doesn't have a cached binary distribution yet (taking version numbers into account). Go to step 4.
4. Install all dependencies from binary distributions based on the list of direct and transitive dependencies obtained in step 1. We have to do these installations manually because easy_install nor pip support binary `*.tar.gz` distributions.

Contact

If you have questions, bug reports, suggestions, etc. please create an issue on the GitHub project page. The latest version of pip-accel will always be available on GitHub. The internal API documentation is hosted on Read The Docs.

License

This software is licensed under the MIT license just like pip (on which pip-accel is based).

CHAPTER 2

Internal API documentation

The second part of the documentation is targeted at developers who wish to extend and/or embed the functionality of `pip-accel`. Here are the contents of the API documentation:

Documentation for the pip accelerator API

On this page you can find the complete API documentation of pip-accel 0.43.

A note about backwards compatibility

Please note that pip-accel has not yet reached a 1.0 version and until that time arbitrary changes to the API can be made. To clarify that statement:

- On the one hand I value API stability and I've built a dozen tools on top of pip-accel myself so I don't think too lightly about breaking backwards compatibility :-)
- On the other hand if I see opportunities to simplify the code base or make things more robust I will go ahead and do it. Furthermore the implementation of pip-accel is dictated (to a certain extent) by pip and this certainly influences the API. For example API changes may be necessary to facilitate the upgrade to pip 1.5.x (the current version of pip-accel is based on pip 1.4.x).

In pip-accel 0.16 a completely new API was introduced and support for the old "API" was dropped. The goal of the new API is to last for quite a while but of course only time will tell if that plan is going to work out :-)

The Python API of pip-accel

Here are the relevant Python modules that make up pip-accel:

- `pip_accel`

`pip_accel`

Top level functionality of *pip-accel*.

The Python module `pip_accel` defines the classes that implement the top level functionality of the pip accelerator. Instead of using the `pip-accel` command you can also use the pip accelerator as a Python module, in this case you'll probably want to start by taking a look at the `PipAccelerator` class.

Wheel support

During the upgrade to pip 6 support for installation of wheels was added to pip-accel. The `pip-accel` command line program now downloads and installs wheels when available for a given requirement, but part of pip-accel's Python API defaults to the more conservative choice of allowing callers to opt-in to wheel support.

This is because previous versions of pip-accel would only download source distributions and pip-accel provides the functionality to convert those source distributions to "dumb binary distributions". This functionality is exposed to callers who may depend on this mode of operation. So for now users of the Python API get to decide whether they're interested in wheels or not.

Setuptools upgrade

If the requirement set includes wheels and `setuptools >= 0.8` is not yet installed, it will be added to the requirement set and installed together with the other requirement(s) in order to enable the usage of distributions installed from wheels (their metadata is different).

class `pip_accel.`**`PipAccelerator`**(*config*, *validate=True*)

Accelerator for pip, the Python package manager.

The *`PipAccelerator`* class brings together the top level logic of pip-accel. This top level logic was previously just a collection of functions but that became more unwieldy as the amount of internal state increased. The *`PipAccelerator`* class is intended to make it (relatively) easy to build something on top of pip and pip-accel.

`__init__`(*config*, *validate=True*)

Initialize the pip accelerator.

Parameters

- **`config`** – The pip-accel configuration (a *`Config`* object).
- **`validate`** – `True` to run *`validate_environment()`*, `False` otherwise.

`validate_environment`()

Make sure `sys.prefix` matches `$VIRTUAL_ENV` (if defined).

This may seem like a strange requirement to dictate but it avoids hairy issues like documented here.

The most sneaky thing is that `pip` doesn't have this problem (de-facto) because `virtualenv` copies `pip` wherever it goes... (`pip-accel` on the other hand has to be installed by the user).

`initialize_directories`()

Automatically create local directories required by pip-accel.

`clean_source_index`()

Cleanup broken symbolic links in the local source distribution index.

The purpose of this method requires some context to understand. Let me preface this by stating that I realize I'm probably overcomplicating things, but I like to preserve forward / backward compatibility when possible and I don't feel like dropping everyone's locally cached source distribution archives without a good reason to do so. With that out of the way:

- Versions of pip-accel based on pip 1.4.x maintained a local source distribution index based on a directory containing symbolic links pointing directly into pip's download cache. When files were removed from pip's download cache, broken symbolic links remained in pip-accel's local source distribution index directory. This resulted in very confusing error messages. To avoid this *`clean_source_index()`* cleaned up broken symbolic links whenever pip-accel was about to invoke pip.
- More recent versions of pip (6.x) no longer support the same style of download cache that contains source distribution archives that can be re-used directly by pip-accel. To cope with the changes in pip 6.x new versions of pip-accel tell pip to download source distribution archives directly into the local source distribution index directory maintained by pip-accel.
- It is very reasonable for users of pip-accel to have multiple versions of pip-accel installed on their system (imagine a dozen Python virtual environments that won't all be updated at the same time; this is the situation I always find myself in :-). These versions of pip-accel will be sharing the same local source distribution index directory.
- All of this leads up to the local source distribution index directory containing a mixture of symbolic links and regular files with no obvious way to atomically and gracefully upgrade the local source distribution index directory while avoiding fights between old and new versions of pip-accel :-).
- I could of course switch to storing the new local source distribution index in a differently named directory (avoiding potential conflicts between multiple versions of pip-accel) but then I would have to introduce a new configuration option, otherwise everyone who has configured pip-accel to store its source index in a non-default location could still be bitten by compatibility issues.

For now I've decided to keep using the same directory for the local source distribution index and to keep cleaning up broken symbolic links. This enables cooperating between old and new versions of pip-accel and avoids trashing user's local source distribution indexes. The main disadvantage is that pip-accel is still required to clean up broken symbolic links...

`install_from_arguments`(*arguments*, ***kw*)

Download, unpack, build and install the specified requirements.

This function is a simple wrapper for *`get_requirements()`*, *`install_requirements()`* and *`cleanup_temporary_directories()`* that implements the default behavior of the pip accelerator. If you're extending or embedding pip-accel you may want to call the underlying methods instead.

If the requirement set includes wheels and `setuptools >= 0.8` is not yet installed, it will be added to the requirement set and installed together with the other requirement(s) in order to enable the usage of distributions installed from wheels (their metadata is different).

Parameters

- **`arguments`** – The command line arguments to `pip install ..` (a list of strings).
- **`kw`** – Any keyword arguments are passed on to *`install_requirements()`*.

Returns The result of *`install_requirements()`*.

`setuptools_supports_wheels`()

Check whether setuptools should be upgraded to `>= 0.8` for wheel support.

Returns `True` when setuptools 0.8 or higher is already installed, `False` otherwise (it needs to be upgraded).

`get_requirements`(*arguments*, *max_retries=None*, *use_wheels=False*)

Use pip to download and unpack the requested source distribution archives.

Parameters

- **`arguments`** – The command line arguments to `pip install ...` (a list of strings).
- **`max_retries`** – The maximum number of times that pip will be asked to download distribution archives (this helps to deal with intermittent failures). If this is `None` then *`max_retries`* is used.
- **`use_wheels`** – Whether pip and pip-accel are allowed to use wheels (`False` by default for backwards compatibility with callers that use pip-accel as a Python API).

> **Warning:** Requirements which are already installed are not included in the result. If this breaks your use case consider using pip's `--ignore-installed` option.

`decorate_arguments`(*arguments*)

Change pathnames of local files into `file://` URLs with `#md5=...` fragments.

Parameters **`arguments`** – The command line arguments to `pip install ...` (a list of strings).

Returns A copy of the command line arguments with pathnames of local files rewritten to `file://` URLs.

When pip-accel calls pip to download missing distribution archives and the user specified the pathname of a local distribution archive on the command line, pip will (by default) *not* copy the archive into the download directory if an archive for the same package name and version is already present.

This can lead to the confusing situation where the user specifies a local distribution archive to install, a different (older) archive for the same package and version is present in the download directory and *pip-accel* installs the older archive instead of the newer archive.

To avoid this confusing behavior, the *`decorate_arguments()`* method rewrites the command line arguments given to `pip install` so that pathnames of local archives are changed into `file://` URLs that include a fragment with the hash of the file's contents. Here's an example:

- Local pathname: `/tmp/pep8-1.6.3a0.tar.gz`
- File URL: `file:///tmp/pep8-1.6.3a0.tar.gz#md5=19cbf0b633498ead63fb3c66e5f1caf6`

When pip fills the download directory and encounters a previously cached distribution archive it will check the hash, realize the contents have changed and replace the archive in the download directory.

`unpack_source_dists`(*arguments*, *use_wheels=False*)

Find and unpack local source distributions and discover their metadata.

Parameters

- **`arguments`** – The command line arguments to `pip install ...` (a list of strings).
- **`use_wheels`** – Whether pip and pip-accel are allowed to use wheels (`False` by default for backwards compatibility with callers that use pip-accel as a Python API).

Returns A list of *`pip_accel.req.Requirement`* objects.

Raises Any exceptions raised by pip, for example `pip.exceptions.DistributionNotFound` when not all requirements can be satisfied.

This function checks whether there are local source distributions available for all requirements, unpacks the source distribution archives and finds the names and versions of the requirements. By using the `pip install --download` command we avoid reimplementing the following pip features:

- Parsing of `requirements.txt` (including recursive parsing).
- Resolution of possibly conflicting pinned requirements.
- Unpacking source distributions in multiple formats.
- Finding the name & version of a given source distribution.

`download_source_dists`(*arguments*, *use_wheels=False*)

Download missing source distributions.

Parameters

- **`arguments`** – The command line arguments to `pip install ...` (a list of strings).
- **`use_wheels`** – Whether pip and pip-accel are allowed to use wheels (`False` by default for backwards compatibility with callers that use pip-accel as a Python API).

Raises Any exceptions raised by pip.

`get_pip_requirement_set`(*arguments*, *use_remote_index*, *use_wheels=False*)

Get the unpacked requirement(s) specified by the caller by running pip.

Parameters

- **`arguments`** – The command line arguments to `pip install ...` (a list of strings).
- **`use_remote_index`** – A boolean indicating whether pip is allowed to connect to the main package index (http://pypi.python.org by default).
- **`use_wheels`** – Whether pip and pip-accel are allowed to use wheels (`False` by default for backwards compatibility with callers that use pip-accel as a Python API).

Returns A `pip.req.RequirementSet` object created by pip.

Raises Any exceptions raised by pip.

`transform_pip_requirement_set`(*requirement_set*)

Transform pip's requirement set into one that *pip-accel* can work with.

Parameters **`requirement_set`** – The `pip.req.RequirementSet` object reported by pip.

Returns A list of *`pip_accel.req.Requirement`* objects.

This function converts the `pip.req.RequirementSet` object reported by pip into a list of *`pip_accel.req.Requirement`* objects.

`install_requirements`(*requirements*, ***kw*)

Manually install a requirement set from binary and/or wheel distributions.

Parameters

- **`requirements`** – A list of *`pip_accel.req.Requirement`* objects.
- **`kw`** – Any keyword arguments are passed on to *`install_binary_dist()`*.

Returns The number of packages that were just installed (an integer).

`arguments_allow_wheels`(*arguments*)

Check whether the given command line arguments allow the use of wheels.

Parameters **`arguments`** – A list of strings with command line arguments.

Returns `True` if the arguments allow wheels, `False` if they disallow wheels.

Contrary to what the name of this method implies its implementation actually checks if the user hasn't *disallowed* the use of wheels using the `--no-use-wheel` option (deprecated in pip 7.x) or the `--no-binary=:all:` option (introduced in pip 7.x). This is because wheels are "opt out" in recent versions of pip. I just didn't like the method name `arguments_dont_disallow_wheels` ;-).

`create_build_directory`()

Create a new build directory for pip to unpack its archives.

`clear_build_directory`()

Clear the build directory where pip unpacks the source distribution archives.

`cleanup_temporary_directories`()

Delete the build directories and any temporary directories created by pip.

`build_directory`

Get the pathname of the current build directory (a string).

class `pip_accel.`**`DownloadLogFilter`**(*name=''*)

Rewrite log messages emitted by pip's `pip.download` module.

When pip encounters hash mismatches it logs a message with the severity `CRITICAL`, however because of the interaction between pip-accel and pip hash mismatches are to be expected and handled gracefully (refer to *`decorate_arguments()`* for details). The *`DownloadLogFilter`* context manager changes the severity of these log messages to `DEBUG` in order to avoid confusing users of pip-accel.

`__enter__`()

Enable the download log filter.

`__exit__`(*exc_type=None*, *exc_value=None*, *traceback=None*)

Disable the download log filter.

filter(*record*)
Change the severity of selected log records.

class `pip_accel.`**SetupRequiresPatch**(*config*, *created_links=None*)
Monkey patch to enable caching of setup requirements.

This context manager monkey patches `InstallRequirement.run_egg_info()` to enable caching of setup requirements. It works by creating a symbolic link called `.eggs` in the source directory of unpacked Python source distributions which points to a shared directory inside the pip-accel data directory. This can only work on platforms that support `os.symlink()`` `()` but should fail gracefully elsewhere.

The *SetupRequiresPatch* context manager doesn't clean up the symbolic links because doing so would remove the link when it is still being used. Instead the context manager builds up a list of created links so that pip-accel can clean these up when it is known that the symbolic links are no longer needed.

For more information about this hack please refer to issue 49.

__init__(*config*, *created_links=None*)
Initialize a *SetupRequiresPatch* object.

Parameters

- **config** – A *Config* object.
- **created_links** – A list where newly created symbolic links are added to (so they can be cleaned up later).

__enter__()
Enable caching of setup requirements (by patching the `run_egg_info()` method).

__exit__(*exc_type=None*, *exc_value=None*, *traceback=None*)
Undo the changes that enable caching of setup requirements.

class `pip_accel.`**CustomPackageFinder**(*find_links*, *index_urls*, *allow_all_prereleases=False*, *trusted_hosts=None*, *process_dependency_links=False*, *session=None*, *format_control=None*, *platform=None*, *versions=None*, *abi=None*, *implementation=None*)
Custom `pip.index.PackageFinder` to keep pip off the internet.

This class customizes `pip.index.PackageFinder` to enforce what the `--no-index` option does for the default package index but doesn't do for package indexes registered with the `--index=` option in requirements files. Judging by pip's documentation the fact that this has to be monkey patched seems like a bug / oversight in pip (IMHO).

index_urls
Dummy list of index URLs that is always empty.

dependency_links
Dummy list of dependency links that is always empty.

class `pip_accel.`**PatchedAttribute**(*object*, *attribute*, *value*, *enabled=True*)
Context manager to temporarily patch an object attribute.

This context manager changes the value of an object attribute when the context is entered and restores the original value when the context is exited.

__init__(*object*, *attribute*, *value*, *enabled=True*)
Initialize a *PatchedAttribute* object.

Parameters

- **object** – The object whose attribute should be patched.
- **attribute** – The name of the attribute to be patched (a string).

- **value** – The temporary value for the attribute.
- **enabled** – True to patch the attribute, False to do nothing instead. This enables conditional attribute patching while unconditionally using the with statement.

__enter__()
Change the object attribute when entering the context.

__exit__(*exc_type=None*, *exc_value=None*, *traceback=None*)
Restore the object attribute when leaving the context.

class pip_accel.**AttributeOverrides**(*opts*, ***overrides*)
AttributeOverrides enables overriding of object attributes.

During the pip 6.x upgrade pip-accel switched to using `pip install --download` which unintentionally broke backwards compatibility with previous versions of pip-accel as documented in issue 52.

The reason for this is that when pip is given the `--download` option it internally enables `--ignore-installed` (which can be problematic for certain use cases as described in issue 52). There is no documented way to avoid this behavior, so instead pip-accel resorts to monkey patching to restore backwards compatibility.

AttributeOverrides is used to replace pip's parsed command line options object with an object that defers all attribute access (gets and sets) to the original options object but always reports `ignore_installed` as False, even after it was set to True by pip (as described above).

__init__(*opts*, ***overrides*)
Construct an *AttributeOverrides* instance.

Parameters

- **opts** – The object to which attribute access is deferred.
- **overrides** – The attributes whose value should be overridden.

__getattr__(*name*)
Get an attribute's value from overrides or by deferring attribute access.

Parameters **name** – The name of the attribute (a string).

Returns The attribute's value.

__setattr__(*name*, *value*)
Set an attribute's value (unless it has an override).

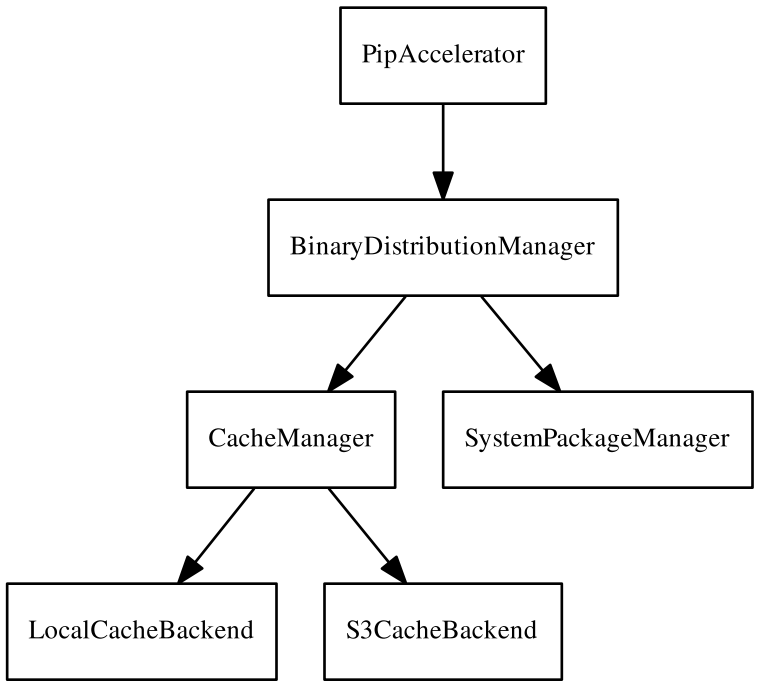
Parameters

- **name** – The name of the attribute (a string).
- **value** – The new value for the attribute.

pip_accel.config

Configuration handling for *pip-accel*.

This module defines the *Config* class which is used throughout the pip accelerator. At runtime an instance of *Config* is created and passed down like this:



The *PipAccelerator* class receives its configuration object from its caller. Usually this will be `main()` but when pip-accel is used as a Python API the person embedding or extending pip-accel is responsible for providing the configuration object. This is intended as a form of dependency injection that enables non-default configurations to be injected into pip-accel.

Support for runtime configuration

The properties of the *Config* class can be set at runtime using regular attribute assignment syntax. This overrides the default values of the properties (whether based on environment variables, configuration files or hard coded defaults).

Support for configuration files

You can use a configuration file to permanently configure certain options of pip-accel. If `/etc/pip-accel.conf` and/or `~/.pip-accel/pip-accel.conf` exist they are automatically loaded. You can also set the environment variable `$PIP_ACCEL_CONFIG` to load a configuration file in a non-default location. If all three files exist the system wide file is loaded first, then the user specific file is loaded and then the file set by the environment variable is loaded (duplicate settings are overridden by the configuration file that's loaded last).

Here is an example of the available options:

```
[pip-accel]
auto-install = yes
max-retries = 3
data-directory = ~/.pip-accel
```

```
s3-bucket = my-shared-pip-accel-binary-cache
s3-prefix = ubuntu-trusty-amd64
s3-readonly = yes
```

Note that the configuration options shown above are just examples, they are not meant to represent the configuration defaults.

---

class `pip_accel.config.`**`Config`**(*load_configuration_files=True*, *load_environment_variables=True*)
Configuration of the pip accelerator.

`__init__`(*load_configuration_files=True*, *load_environment_variables=True*)
Initialize the configuration of the pip accelerator.

Parameters

- **`load_configuration_files`** – If this is `True` (the default) then configuration files in known locations are automatically loaded.
- **`load_environment_variables`** – If this is `True` (the default) then environment variables are used to initialize the configuration.

`available_configuration_files`
A list of strings with the absolute pathnames of the available configuration files.

`load_configuration_file`(*configuration_file*)
Load configuration defaults from a configuration file.

Parameters **`configuration_file`** – The pathname of a configuration file (a string).

Raises `Exception` when the configuration file cannot be loaded.

`__setattr__`(*name*, *value*)
Override the value of a property at runtime.

Parameters

- **`name`** – The name of the property to override (a string).
- **`value`** – The overridden value of the property.

`get`(*property_name=None*, *environment_variable=None*, *configuration_option=None*, *default=None*)
Internal shortcut to get a configuration option's value.

Parameters

- **`property_name`** – The name of the property that users can set on the *`Config`* class (a string).
- **`environment_variable`** – The name of the environment variable (a string).
- **`configuration_option`** – The name of the option in the configuration file (a string).
- **`default`** – The default value.

Returns The value of the environment variable or configuration file option or the default value.

`cache_format_revision`
The revision of the binary distribution cache format in use (an integer).

This number is encoded in the directory name of the binary cache so that multiple revisions can peacefully coexist. When pip-accel breaks backwards compatibility this number is bumped so that pip-accel starts using a new directory.

source_index
The absolute pathname of pip-accel's source index directory (a string).

This is the `sources` subdirectory of *data_directory*.

binary_cache
The absolute pathname of pip-accel's binary cache directory (a string).

This is the `binaries` subdirectory of *data_directory*.

eggs_cache
The absolute pathname of pip-accel's eggs cache directory (a string).

This is the `eggs` subdirectory of *data_directory*. It is used to cache setup requirements which avoids continuous rebuilding of setup requirements.

data_directory
The absolute pathname of the directory where pip-accel's data files are stored (a string).

- Environment variable: `$PIP_ACCEL_CACHE`
- Configuration option: `data-directory`
- Default: `/var/cache/pip-accel` if running as `root`, `~/.pip-accel` otherwise

on_debian
`True` if running on a Debian derived system, `False` otherwise.

install_prefix
The absolute pathname of the installation prefix to use (a string).

This property is based on `sys.prefix` except that when `sys.prefix` is `/usr` and we're running on a Debian derived system `/usr/local` is used instead.

The reason for this is that on Debian derived systems only apt (dpkg) should be allowed to touch files in `/usr/lib/pythonX.Y/dist-packages` and `python setup.py install` knows this (see the `posix_local` installation scheme in `/usr/lib/pythonX.Y/sysconfig.py` on Debian derived systems). Because pip-accel replaces `python setup.py install` it has to replicate this logic. Inferring all of this from the `sysconfig` module would be nice but that module wasn't available in Python 2.6.

python_executable
The absolute pathname of the Python executable (a string).

auto_install
Whether automatic installation of missing system packages is enabled.

`True` if automatic installation of missing system packages is enabled, `False` if it is disabled, `None` otherwise (in this case the user will be prompted at the appropriate time).

- Environment variable: `$PIP_ACCEL_AUTO_INSTALL` (refer to `coerce_boolean()` for details on how the value of the environment variable is interpreted)
- Configuration option: `auto-install` (also parsed using `coerce_boolean()`)
- Default: `None`

log_format
The format of log messages written to the terminal.

- Environment variable: `$PIP_ACCEL_LOG_FORMAT`
- Configuration option: `log-format`
- Default: `DEFAULT_LOG_FORMAT`

log_verbosity

The verbosity of log messages written to the terminal.

- Environment variable: `$PIP_ACCEL_LOG_VERBOSITY`
- Configuration option: `log-verbosity`
- Default: 'INFO' (a string).

max_retries

The number of times to retry `pip install --download` if it fails.

- Environment variable: `$PIP_ACCEL_MAX_RETRIES`
- Configuration option: `max-retries`
- Default: `3`

trust_mod_times

Whether to trust file modification times for cache invalidation.

- Environment variable: `$PIP_ACCEL_TRUST_MOD_TIMES`
- Configuration option: `trust-mod-times`
- **Default: `True` unless the AppVeyor continuous integration** environment is detected (see issue 62).

s3_cache_url

The URL of the Amazon S3 API endpoint to use.

By default this points to the official Amazon S3 API endpoint. You can change this option if you're running a local Amazon S3 compatible storage service that you want pip-accel to use.

- Environment variable: `$PIP_ACCEL_S3_URL`
- Configuration option: `s3-url`
- Default: `https://s3.amazonaws.com`

For details please refer to the *`pip_accel.caches.s3`* module.

s3_cache_bucket

Name of Amazon S3 bucket where binary distributions are cached (a string or `None`).

- Environment variable: `$PIP_ACCEL_S3_BUCKET`
- Configuration option: `s3-bucket`
- Default: `None`

For details please refer to the *`pip_accel.caches.s3`* module.

s3_cache_create_bucket

Whether to automatically create the Amazon S3 bucket when it's missing.

- Environment variable: `$PIP_ACCEL_S3_CREATE_BUCKET`
- Configuration option: `s3-create-bucket`
- Default: `False`

For details please refer to the *`pip_accel.caches.s3`* module.

s3_cache_prefix

Cache prefix for binary distribution archives in Amazon S3 bucket (a string or `None`).

- Environment variable: `$PIP_ACCEL_S3_PREFIX`

- Configuration option: `s3-prefix`
- Default: `None`

For details please refer to the *`pip_accel.caches.s3`* module.

`s3_cache_readonly`
Whether the Amazon S3 bucket is considered read only.

If this is `True` then the Amazon S3 bucket will only be used for *`get()`* operations (all *`put()`* operations will be disabled).

- Environment variable: `$PIP_ACCEL_S3_READONLY` (refer to `coerce_boolean()` for details on how the value of the environment variable is interpreted)
- Configuration option: `s3-readonly` (also parsed using `coerce_boolean()`)
- Default: `False`

For details please refer to the *`pip_accel.caches.s3`* module.

`s3_cache_timeout`
The socket timeout in seconds for connections to Amazon S3 (an integer).

This value is injected into Boto's configuration to override the default socket timeout used for connections to Amazon S3.

- Environment variable: `$PIP_ACCEL_S3_TIMEOUT`
- Configuration option: `s3-timeout`
- Default: `60` (Boto's default)

`s3_cache_retries`
The number of times to retry failed requests to Amazon S3 (an integer).

This value is injected into Boto's configuration to override the default number of times to retry failed requests to Amazon S3.

- Environment variable: `$PIP_ACCEL_S3_RETRIES`
- Configuration option: `s3-retries`
- Default: `5` (Boto's default)

`pip_accel.req`

Simple wrapper for pip and pkg_resources *Requirement* objects.

After downloading the specified requirement(s) pip reports a "requirement set" to pip-accel. In the past pip-accel would summarize this requirement set into a list of tuples, where each tuple would contain a requirement's project name, version and source directory (basically only the information required by pip-accel remained).

Recently I've started using pip-accel as a library in another project I'm working on (not yet public) and in that project I am very interested in whether a given requirement is a direct or transitive requirement. Unfortunately pip-accel did not preserve this information.

That's when I decided that next to pip's `pip.req.InstallRequirement` and setuptools' `pkg_resources.Requirement` I would introduce yet another type of requirement object... It's basically just a summary of the other two types of requirement objects and it also provides access to the original requirement objects (for those who are interested; the interfaces are basically undocumented AFAIK).

class `pip_accel.req.`**`Requirement`**(*config*, *requirement*)
Simple wrapper for the requirement objects defined by pip and setuptools.

`__init__`(*config*, *requirement*)
Initialize a requirement object.

Parameters

- **`config`** – A *`Config`* object.
- **`requirement`** – A `pip.req.InstallRequirement` object.

`__repr__`()
Generate a human friendly representation of a requirement object.

`name`
The name of the Python package (a string).

This is the name used to register a package on PyPI and the name reported by commands like `pip freeze`. Based on `pkg_resources.Requirement.project_name`.

`version`
The version of the package that `pip` wants to install (a string).

`related_archives`
The pathnames of the source distribution(s) for this requirement (a list of strings).

Note: This property is very new in pip-accel and its logic may need some time to mature. For now any misbehavior by this property shouldn't be too much of a problem because the pathnames reported by this property are only used for cache invalidation (see the *`last_modified`* and *`checksum`* properties).

`last_modified`
The last modified time of the requirement's source distribution archive(s) (a number).

The value of this property is based on the *`related_archives`* property. If no related archives are found the current time is reported. In the balance between not invalidating cached binary distributions enough and invalidating them too frequently, this property causes the latter to happen.

`checksum`
The SHA1 checksum of the requirement's source distribution archive(s) (a string).

The value of this property is based on the *`related_archives`* property. If no related archives are found the SHA1 digest of the empty string is reported.

`source_directory`
The pathname of the directory containing the unpacked source distribution (a string).

This is the directory that contains a `setup.py` script. Based on `pip.req.InstallRequirement.source_dir`.

`is_wheel`
`True` when the requirement is a wheel, `False` otherwise.

Note: To my surprise it seems to be non-trivial to determine whether a given `pip.req.InstallRequirement` object produced by pip's internal Python API concerns a source distribution or a wheel distribution.

There's a `pip.req.InstallRequirement.is_wheel` property but I'm currently looking at a wheel distribution whose `is_wheel` property returns `None`, apparently because the requirement's `url` property is also `None`.

Whether this is an obscure implementation detail of pip or caused by the way pip-accel invokes pip, I really can't tell (yet).

---

is_transitive
Whether the dependency is transitive (indirect).

`True` when the requirement is a transitive dependency (a dependency of a dependency) or `False` when the requirement is a direct dependency (specified on pip's command line or in a `requirements.txt` file). Based on `pip.req.InstallRequirement.comes_from`.

is_direct
The opposite of *`Requirement.is_transitive`*.

is_editable
Whether the requirement should be installed in editable mode.

`True` when the requirement is to be installed in editable mode (i.e. setuptools "develop mode"). Based on `pip.req.InstallRequirement.editable`.

sdist_metadata
Get the distribution metadata of an unpacked source distribution.

wheel_metadata
Get the distribution metadata of an unpacked wheel distribution.

__str__()
Render a human friendly string describing the requirement.

class `pip_accel.req.`**TransactionalUpdate**(*requirement*)
Context manager that enables transactional package upgrades.

__init__(*requirement*)
Initialize a *`TransactionalUpdate`* object.

Parameters **requirement** – A *`Requirement`* object.

__enter__()
Prepare package upgrades by removing conflicting installations.

__exit__(*exc_type=None*, *exc_value=None*, *traceback=None*)
Finalize or rollback a package upgrade.

`pip_accel.req.`**escape_name**(*requirement_name*)
Escape a requirement's name for use in a regular expression.

This backslash-escapes all non-alphanumeric characters and replaces dashes and underscores with a character class that matches a dash or underscore (effectively treating dashes and underscores equivalently).

Parameters **requirement_name** – The name of the requirement (a string).

Returns The requirement's name as a regular expression (a string).

`pip_accel.req.`**escape_name_callback**(*match*)
Used by *`escape_name()`* to treat dashes and underscores as equivalent.

Parameters **match** – A regular expression match object that captured a single character.

Returns A regular expression string that matches the captured character.

pip_accel.bdist

Functions to manipulate Python binary distribution archives.

The functions in this module are used to create, transform and install from binary distribution archives (which are not supported by tools like easy_install and pip).

class `pip_accel.bdist.`**`BinaryDistributionManager`**(*config*)

Generates and transforms Python binary distributions.

`__init__`(*config*)

Initialize the binary distribution manager.

Parameters **`config`** – The pip-accel configuration (a *`Config`* object).

`get_binary_dist`(*requirement*)

Get or create a cached binary distribution archive.

Parameters **`requirement`** – A *`Requirement`* object.

Returns An iterable of tuples with two values each: A `tarfile.TarInfo` object and a file-like object.

Gets the cached binary distribution that was previously built for the given requirement. If no binary distribution has been cached yet, a new binary distribution is built and added to the cache.

Uses *`build_binary_dist()`* to build binary distribution archives. If this fails with a build error *`get_binary_dist()`* will use *`SystemPackageManager`* to check for and install missing system packages and retry the build when missing system packages were installed.

`needs_invalidation`(*requirement*, *cache_file*)

Check whether a cached binary distribution needs to be invalidated.

Parameters

- **`requirement`** – A *`Requirement`* object.
- **`cache_file`** – The pathname of a cached binary distribution (a string).

Returns `True` if the cached binary distribution needs to be invalidated, `False` otherwise.

`recall_checksum`(*cache_file*)

Get the checksum of the input used to generate a binary distribution archive.

Parameters **`cache_file`** – The pathname of the binary distribution archive (a string).

Returns The checksum (a string) or `None` (when no checksum is available).

`persist_checksum`(*requirement*, *cache_file*)

Persist the checksum of the input used to generate a binary distribution.

Parameters

- **`requirement`** – A *`Requirement`* object.
- **`cache_file`** – The pathname of a cached binary distribution (a string).

Note: The checksum is only calculated and persisted when *`trust_mod_times`* is `False`.

`build_binary_dist`(*requirement*)

Build a binary distribution archive from an unpacked source distribution.

Parameters **`requirement`** – A *`Requirement`* object.

Returns The pathname of a binary distribution archive (a string).

Raises *`BinaryDistributionError`* when the original command and the fall back both fail to produce a binary distribution archive.

This method uses the following command to build binary distributions:

```
$ python setup.py bdist_dumb --format=tar
```

This command can fail for two main reasons:

1. The package is missing binary dependencies.
2. The `setup.py` script doesn't (properly) implement `bdist_dumb` binary distribution format support.

The first case is dealt with in *`get_binary_dist()`*. To deal with the second case this method falls back to the following command:

```
$ python setup.py bdist
```

This fall back is almost never needed, but there are Python packages out there which require this fall back (this method was added because the installation of `Paver==1.2.3` failed, see issue 37 for details about that).

`build_binary_dist_helper`(*requirement*, *setup_command*)

Convert an unpacked source distribution to a binary distribution.

Parameters

- **`requirement`** – A *`Requirement`* object.
- **`setup_command`** – A list of strings with the arguments to `setup.py`.

Returns The pathname of the resulting binary distribution (a string).

Raises *`BuildFailed`* when the build reports an error (e.g. because of missing binary dependencies like system libraries).

Raises *`NoBuildOutput`* when the build does not produce the expected binary distribution archive.

`transform_binary_dist`(*archive_path*)

Transform binary distributions into a form that can be cached for future use.

Parameters **`archive_path`** – The pathname of the original binary distribution archive.

Returns

An iterable of tuples with two values each:

1. A `tarfile.TarInfo` object.
2. A file-like object.

This method transforms a binary distribution archive created by *`build_binary_dist()`* into a form that can be cached for future use. This comes down to making the pathnames inside the archive relative to the *prefix* that the binary distribution was built for.

`install_binary_dist`(*members*, *virtualenv_compatible=True*, *prefix=None*, *python=None*, *track_installed_files=False*)

Install a binary distribution into the given prefix.

Parameters

- **`members`** – An iterable of tuples with two values each:
  1. A `tarfile.TarInfo` object.
  2. A file-like object.

- **prefix** – The "prefix" under which the requirements should be installed. This will be a pathname like `/usr`, `/usr/local` or the pathname of a virtual environment. Defaults to *Config.install_prefix*.
- **python** – The pathname of the Python executable to use in the shebang line of all executable Python scripts inside the binary distribution. Defaults to *Config.python_executable*.
- **virtualenv_compatible** – Whether to enable workarounds to make the resulting filenames compatible with virtual environments (defaults to `True`).
- **track_installed_files** – If this is `True` (not the default for this method because of backwards compatibility) pip-accel will create `installed-files.txt` as required by pip to properly uninstall packages.

This method installs a binary distribution created by *build_binary_dist()* into the given prefix (a directory like `/usr`, `/usr/local` or a virtual environment).

fix_hashbang(*contents*, *python*)

Rewrite hashbangs to use the correct Python executable.

Parameters

- **contents** – The contents of the script whose hashbang should be fixed (a string).
- **python** – The absolute pathname of the Python executable (a string).

Returns The modified contents of the script (a string).

update_installed_files(*installed_files*)

Track the files installed by a package so pip knows how to remove the package.

This method is used by *install_binary_dist()* (which collects the list of installed files for *update_installed_files()*).

Parameters installed_files – A list of absolute pathnames (strings) with the files that were just installed.

pip_accel.caches

Support for multiple cache backends.

This module defines an abstract base class (*AbstractCacheBackend*) to be inherited by custom cache backends in order to easily integrate them in pip-accel. The cache backends included in pip-accel are built on top of the same mechanism.

Additionally this module defines *CacheManager* which makes it possible to merge the available cache backends into a single logical cache which automatically disables backends that report errors.

class `pip_accel.caches.`**CacheBackendMeta**(*name*, *bases*, *dict*)

Metaclass to intercept cache backend definitions.

__init__(*name*, *bases*, *dict*)

Intercept cache backend definitions.

class `pip_accel.caches.`**AbstractCacheBackend**(*config*)

Abstract base class for implementations of pip-accel cache backends.

Subclasses of this class are used by pip-accel to store Python distribution archives in order to accelerate performance and gain independence of external systems like PyPI and distribution sites.

Note: This base class automatically registers subclasses at definition time, providing a simple and elegant registration mechanism for custom backends. This technique uses metaclasses and was originally based on the article Using Metaclasses to Create Self-Registering Plugins.

I've since had to introduce some additional magic to make this mechanism compatible with both Python 2.x and Python 3.x because the syntax for metaclasses is very much incompatible and I refuse to write separate implementations for both :-).

`__init__`(*config*)
Initialize a cache backend.

Parameters **`config`** – The pip-accel configuration (a *`Config`* object).

`get`(*filename*)
Get a previously cached distribution archive from the cache.

Parameters **`filename`** – The expected filename of the distribution archive (a string).

Returns The absolute pathname of a local file or `None` when the distribution archive hasn't been cached.

This method is called by *pip-accel* before fetching or building a distribution archive, in order to check whether a previously cached distribution archive is available for re-use.

`put`(*filename*, *handle*)
Store a newly built distribution archive in the cache.

Parameters

- **`filename`** – The filename of the distribution archive (a string).
- **`handle`** – A file-like object that provides access to the distribution archive.

This method is called by *pip-accel* after fetching or building a distribution archive, in order to cache the distribution archive.

`__repr__`()
Generate a textual representation of the cache backend.

class `pip_accel.caches.`**`CacheManager`**(*config*)
Interface to treat multiple cache backends as a single one.

The cache manager automatically disables cache backends that raise exceptions on `get()` and `put()` operations.

`__init__`(*config*)
Initialize a cache manager.

Automatically initializes instances of all registered cache backends based on setuptools' support for entry points which makes it possible for external Python packages to register additional cache backends without any modifications to pip-accel.

Parameters **`config`** – The pip-accel configuration (a *`Config`* object).

`get`(*requirement*)
Get a distribution archive from any of the available caches.

Parameters **`requirement`** – A *`Requirement`* object.

Returns The absolute pathname of a local file or `None` when the distribution archive is missing from all available caches.

put (*requirement*, *handle*)
Store a distribution archive in all of the available caches.

Parameters

- **requirement** – A `Requirement` object.
- **handle** – A file-like object that provides access to the distribution archive.

generate_filename (*requirement*)
Generate a distribution archive filename for a package.

Parameters **requirement** – A `Requirement` object.

Returns The filename of the distribution archive (a string) including a single leading directory component to indicate the cache format revision.

pip_accel.caches.local

Local file system cache backend.

This module implements the local cache backend which stores distribution archives on the local file system. This is a very simple cache backend, all it does is create directories and write local files. The only trick here is that new binary distribution archives are written to temporary files which are then moved into place atomically using `os.rename()` to avoid partial reads caused by running multiple invocations of pip-accel at the same time (which happened in issue 25).

class `pip_accel.caches.local.`**LocalCacheBackend** (*config*)
The local cache backend stores Python distribution archives on the local file system.

get (*filename*)
Check if a distribution archive exists in the local cache.

Parameters **filename** – The filename of the distribution archive (a string).

Returns The pathname of a distribution archive on the local file system or `None`.

put (*filename*, *handle*)
Store a distribution archive in the local cache.

Parameters

- **filename** – The filename of the distribution archive (a string).
- **handle** – A file-like object that provides access to the distribution archive.

pip_accel.caches.s3

Amazon S3 cache backend.

This module implements a cache backend that stores distribution archives in a user defined Amazon S3 bucket. To enable this backend you need to define the configuration option `s3_cache_bucket` and configure your Amazon S3 API credentials (see the readme for details).

Using S3 compatible storage services

The Amazon S3 API has been implemented in several open source projects and dozens of online services. To use pip-accel with an S3 compatible storage service you can override the `s3_cache_url` option. The pip-accel test suite actually uses this option to test the S3 cache backend by running FakeS3 in the background and pointing pip-accel at the FakeS3 server. Below are some usage notes that may be relevant for people evaluating this option.

Secure connections Boto has to be told whether to make a "secure" connection to the S3 API and pip-accel assumes the `https://` URL scheme implies a secure connection while the `http://` URL scheme implies a non-secure connection.

Calling formats Boto has the concept of "calling formats" for the S3 API and to connect to the official Amazon S3 API pip-accel needs to specify the "sub-domain calling format" or the API calls will fail. When you specify a nonstandard S3 API URL pip-accel tells Boto to use the "ordinary calling format" instead. This differentiation will undoubtedly not be correct in all cases. If this is bothering you then feel free to open an issue on GitHub to make pip-accel more flexible in this regard.

Credentials If you don't specify S3 API credentials and the connection attempt to S3 fails with "NoAuthHandlerFound: No handler was ready to authenticate" pip-accel will fall back to an anonymous connection attempt. If that fails as well the S3 cache backend is disabled. It may be useful to note here that the pip-accel test suite uses FakeS3 and the anonymous connection fall back works fine.

A note about robustness

The Amazon S3 cache backend implemented in *`pip_accel.caches.s3`* is specifically written to gracefully disable itself when it encounters known errors such as:

- The configuration option *`s3_cache_bucket`* is not set (i.e. the user hasn't configured the backend yet).
- The `boto` package is not installed (i.e. the user ran `pip install pip-accel` instead of `pip install 'pip-accel[s3]'`).
- The connection to the S3 API can't be established (e.g. because API credentials haven't been correctly configured).
- The connection to the configured S3 bucket can't be established (e.g. because the bucket doesn't exist or the configured credentials don't provide access to the bucket).

Additionally *`CacheManager`* automatically disables cache backends that raise exceptions on *`get()`* and *`put()`* operations. The end result is that when the S3 backend fails you will just revert to using the cache on the local file system.

Optionally if you are using read only credentials you can disable *`put()`* operations by setting the configuration option *`s3_cache_readonly`*.

---

class `pip_accel.caches.s3.`**`S3CacheBackend`**(*config*)

The S3 cache backend stores distribution archives in a user defined Amazon S3 bucket.

`get`(*filename*)

Download a distribution archive from the configured Amazon S3 bucket.

Parameters **`filename`** – The filename of the distribution archive (a string).

Returns The pathname of a distribution archive on the local file system or `None`.

Raises *`CacheBackendError`* when any underlying method fails.

`put`(*filename*, *handle*)

Upload a distribution archive to the configured Amazon S3 bucket.

If the *`s3_cache_readonly`* configuration option is enabled this method does nothing.

Parameters

- **`filename`** – The filename of the distribution archive (a string).
- **`handle`** – A file-like object that provides access to the distribution archive.

Raises *CacheBackendError* when any underlying method fails.

s3_bucket
Connect to the user defined Amazon S3 bucket.

Called on demand by *get()* and *put()*. Caches its return value so that only a single connection is created.

Returns A `boto.s3.bucket.Bucket` object.

Raises *CacheBackendDisabledError* when the user hasn't defined *Config.s3_cache_bucket*.

Raises *CacheBackendError* when the connection to the Amazon S3 bucket fails.

s3_connection
Connect to the Amazon S3 API.

If the connection attempt fails because Boto can't find credentials the attempt is retried once with an anonymous connection.

Called on demand by *s3_bucket*.

Returns A `boto.s3.connection.S3Connection` object.

Raises *CacheBackendError* when the connection to the Amazon S3 API fails.

get_cache_key(*filename*)
Compose an S3 cache key based on *Config.s3_cache_prefix* and the given filename.

Parameters **filename** – The filename of the distribution archive (a string).

Returns The cache key for the given filename (a string).

check_prerequisites()
Validate the prerequisites required to use the Amazon S3 cache backend.

Makes sure the Amazon S3 cache backend is configured (*Config.s3_cache_bucket* is defined by the user) and `boto` is available for use.

Raises *CacheBackendDisabledError* when a prerequisite fails.

class `pip_accel.caches.s3.`**PatchedBotoConfig**
Monkey patch for Boto's configuration handling.

Boto's configuration handling is kind of broken on Python 3 as documented here. The *PatchedBotoConfig* class implements a context manager that temporarily patches Boto to work around the bug.

Without this monkey patch it is impossible to configure the number of retries on Python 3 which makes the pip-accel test suite horribly slow.

__init__()
Initialize a *PatchedBotoConfig* object.

get(*section*, *name*, *default=None*, ***kw*)
Replacement for `boto.pyami.config.Config.get()`.

pip_accel.deps

System package dependency handling.

The *pip_accel.deps* module is an extension of pip-accel that deals with dependencies on system packages. Currently only Debian Linux and derivative Linux distributions are supported by this extension but it should be fairly easy to add support for other platforms.

The interface between pip-accel and `SystemPackageManager` focuses on `install_dependencies()` (the other methods are used internally).

class `pip_accel.deps.`**`SystemPackageManager`**(*config*)

Interface to the system's package manager.

`__init__`(*config*)

Initialize the system package dependency manager.

Parameters **`config`** – The pip-accel configuration (a `Config` object).

`install_dependencies`(*requirement*)

Install missing dependencies for the given requirement.

Parameters **`requirement`** – A `Requirement` object.

Returns `True` when missing system packages were installed, `False` otherwise.

Raises `DependencyInstallationRefused` when automatic installation is disabled or refused by the operator.

Raises `DependencyInstallationFailed` when the installation of missing system packages fails.

If *pip-accel* fails to build a binary distribution, it will call this method as a last chance to install missing dependencies. If this function does not raise an exception, *pip-accel* will retry the build once.

`find_missing_dependencies`(*requirement*)

Find missing dependencies of a Python package.

Parameters **`requirement`** – A `Requirement` object.

Returns A list of strings with system package names.

`find_known_dependencies`(*requirement*)

Find the known dependencies of a Python package.

Parameters **`requirement`** – A `Requirement` object.

Returns A list of strings with system package names.

`find_installed_packages`()

Find the installed system packages.

Returns A list of strings with system package names.

Raises `SystemDependencyError` when the command to list the installed system packages fails.

`installation_refused`(*requirement*, *missing_dependencies*, *reason*)

Raise `DependencyInstallationRefused` with a user friendly message.

Parameters

- **`requirement`** – A `Requirement` object.
- **`missing_dependencies`** – A list of strings with missing dependencies.
- **`reason`** – The reason why installation was refused (a string).

`confirm_installation`(*requirement*, *missing_dependencies*, *install_command*)

Ask the operator's permission to install missing system packages.

Parameters

- **`requirement`** – A `Requirement` object.

- **missing_dependencies** – A list of strings with missing dependencies.
- **install_command** – A list of strings with the command line needed to install the missing dependencies.

Raises *DependencyInstallationRefused* when the operator refuses.

pip_accel.utils

Utility functions for the pip accelerator.

The *pip_accel.utils* module defines several miscellaneous/utility functions that are used throughout *pip_accel* but don't really belong with any single module.

pip_accel.utils.**compact**(*text*, ***kw*)

Compact whitespace in a string and format any keyword arguments into the string.

Parameters

- **text** – The text to compact (a string).
- **kw** – Any keyword arguments to apply using `str.format()`.

Returns The compacted, formatted string.

The whitespace compaction preserves paragraphs.

pip_accel.utils.**expand_path**(*pathname*)

Expand the home directory in a pathname based on the effective user id.

Parameters **pathname** – A pathname that may start with `~/`, indicating the path should be interpreted as being relative to the home directory of the current (effective) user.

Returns The (modified) pathname.

This function is a variant of `os.path.expanduser()` that doesn't use `$HOME` but instead uses the home directory of the effective user id. This is basically a workaround for `sudo -s` not resetting `$HOME`.

pip_accel.utils.**create_file_url**(*pathname*)

Create a `file:...` URL from a local pathname.

Parameters **pathname** – The pathname of a local file or directory (a string).

Returns A URL that refers to the local file or directory (a string).

pip_accel.utils.**find_home_directory**()

Look up the home directory of the effective user id.

Returns The pathname of the home directory (a string).

Note: On Windows this uses the `%APPDATA%` environment variable (if available) and otherwise falls back to `~/Application Data`.

pip_accel.utils.**is_root**()

Detect whether we're running with super user privileges.

pip_accel.utils.**get_python_version**()

Get a string identifying the currently running Python version.

This function generates a string that uniquely identifies the currently running Python implementation and version. The Python implementation is discovered using `platform.python_implementation()` and the major and minor version numbers are extracted from `sys.version_info`.

Returns A string containing the name of the Python implementation and the major and minor version numbers.

Example:

```
>>> from pip_accel.utils import get_python_version
>>> get_python_version()
'CPython-2.7'
```

`pip_accel.utils.`**`makedirs`**`(`*path*, *mode=511*`)`
Create a directory if it doesn't already exist (keeping concurrency in mind).

Parameters

- **`path`** – The pathname of the directory to create (a string).
- **`mode`** – The mode to apply to newly created directories (an integer, defaults to the octal number `0777`).

Returns `True` when the directory was created, `False` if it already existed.

Raises Any exceptions raised by `os.makedirs()` except for `errno.EEXIST` (this error is swallowed and `False` is returned instead).

`pip_accel.utils.`**`same_directories`**`(`*path1*, *path2*`)`
Check if two pathnames refer to the same directory.

Parameters

- **`path1`** – The first pathname (a string).
- **`path2`** – The second pathname (a string).

Returns `True` if both pathnames refer to the same directory, `False` otherwise.

`pip_accel.utils.`**`hash_files`**`(`*method*, **files*`)`
Calculate the hexadecimal digest of one or more local files.

Parameters

- **`method`** – The hash method (a string, given to `hashlib.new()`).
- **`files`** – The pathname(s) of file(s) to hash (zero or more strings).

Returns The calculated hex digest (a string).

`pip_accel.utils.`**`replace_file`**`(`*src*, *dst*`)`
Overwrite a file (in an atomic fashion when possible).

Parameters

- **`src`** – The pathname of the source file (a string).
- **`dst`** – The pathname of the destination file (a string).

class `pip_accel.utils.`**`AtomicReplace`**`(`*filename*`)`
Context manager to atomically replace a file's contents.

`__init__``(`*filename*`)`
Initialize a *`AtomicReplace`* object.

Parameters **`filename`** – The pathname of the file to replace (a string).

`__enter__``()`
Prepare to replace the file's contents.

Returns The pathname of a temporary file in the same directory as the file to replace (a string). Using this temporary file ensures that *`replace_file()`* doesn't fail due to a cross-device rename operation.

`__exit__`(*exc_type=None*, *exc_value=None*, *traceback=None*)
Replace the file's contents (if no exception occurred) using *`replace_file()`*.

`pip_accel.utils.`**`requirement_is_installed`**(*expr*)
Check whether a requirement is installed.

Parameters **`expr`** – A requirement specification similar to those used in pip requirement files (a string).

Returns `True` if the requirement is available (installed), `False` otherwise.

`pip_accel.utils.`**`is_installed`**(*package_name*)
Check whether a package is installed in the current environment.

Parameters **`package_name`** – The name of the package (a string).

Returns `True` if the package is installed, `False` otherwise.

`pip_accel.utils.`**`uninstall`**(**package_names*)
Uninstall one or more packages using the Python equivalent of `pip uninstall --yes`.

The package(s) to uninstall must be installed, otherwise pip will raise an `UninstallationError`. You can check for installed packages using *`is_installed()`*.

Parameters **`package_names`** – The names of one or more Python packages (strings).

`pip_accel.utils.`**`match_option`**(*argument*, *short_option*, *long_option*)
Match a command line argument against a short and long option.

Parameters

- **`argument`** – The command line argument (a string).
- **`short_option`** – The short option (a string).
- **`long_option`** – The long option (a string).

Returns `True` if the argument matches, `False` otherwise.

`pip_accel.utils.`**`is_short_option`**(*argument*)
Check if a command line argument is a short option.

Parameters **`argument`** – The command line argument (a string).

Returns `True` if the argument is a short option, `False` otherwise.

`pip_accel.utils.`**`match_option_with_value`**(*arguments*, *option*, *value*)
Check if a list of command line options contains an option with a value.

Parameters

- **`arguments`** – The command line arguments (a list of strings).
- **`option`** – The long option (a string).
- **`value`** – The expected value (a string).

Returns `True` if the command line contains the option/value pair, `False` otherwise.

`pip_accel.utils.`**`contains_sublist`**(*lst*, *sublst*)
Check if one list contains the items from another list (in the same order).

Parameters

- **`lst`** – The main list.
- **`sublist`** – The sublist to check for.

Returns `True` if the main list contains the items from the sublist in the same order, `False` otherwise.

Based on this StackOverflow answer.

`pip_accel.exceptions`

Exceptions for structured error handling.

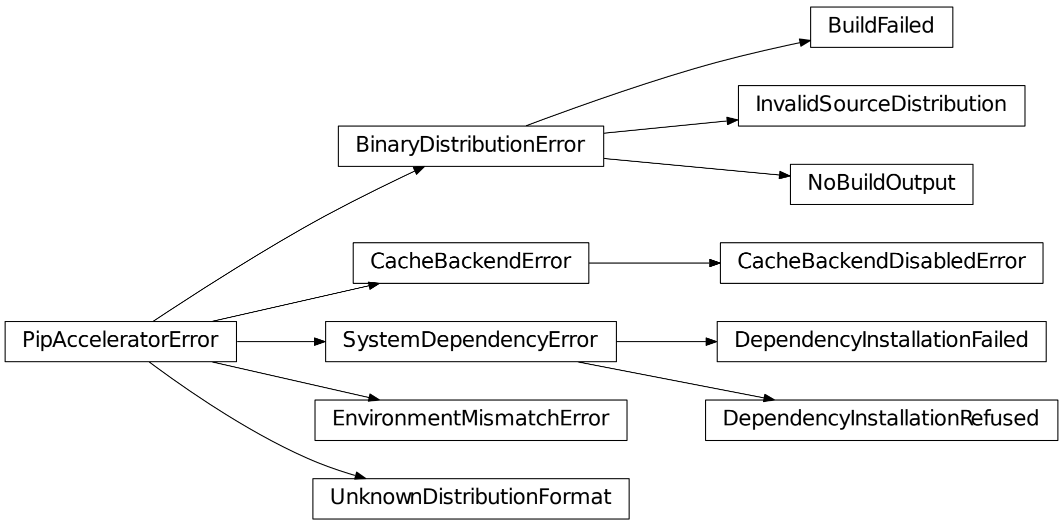
This module defines named exceptions raised by pip-accel when it encounters error conditions that:

1. Already require structured handling inside pip-accel
2. May require structured handling by callers of pip-accel

Yes, I know, I just made your lovely and elegant Python look a whole lot like Java! I guess the message to take away here is that (in my opinion) structured error handling helps to build robust software that acknowledges failures exist and tries to deal with them (even if only by clearly recognizing a problem and giving up when there's nothing useful to do!).

Hierarchy of exceptions

If you're interested in implementing structured handling of exceptions reported by pip-accel the following diagram may help by visualizing the hierarchy:


BuildFailed
InvalidSourceDistribution
BinaryDistributionError
NoBuildOutput
CacheBackendError
CacheBackendDisabledError
PipAcceleratorError
SystemDependencyError
DependencyInstallationFailed
EnvironmentMismatchError
DependencyInstallationRefused
UnknownDistributionFormat

exception `pip_accel.exceptions.`**`PipAcceleratorError`**(*text*, ***kw*)

Base exception for all exception types explicitly raised by *`pip_accel`*.

`__init__`(*text*, ***kw*)
Initialize a `PipAcceleratorError` object.

Accepts the same arguments as `compact()`.

exception `pip_accel.exceptions.`**`NothingToDoError`**(*text*, ***kw*)
Custom exception raised on empty requirement sets.

Raised by `get_pip_requirement_set()` when pip doesn't report an error but also doesn't generate a requirement set (this happens when the user specifies an empty requirements file).

exception `pip_accel.exceptions.`**`EnvironmentMismatchError`**(*text*, ***kw*)
Custom exception raised when a cross-environment action is attempted.

Raised by `validate_environment()` when it detects a mismatch between `sys.prefix` and `$VIRTUAL_ENV`.

exception `pip_accel.exceptions.`**`UnknownDistributionFormat`**(*text*, ***kw*)
Custom exception raised on unrecognized distribution archives.

Raised by `is_wheel` when it cannot discern whether a given unpacked distribution is a source distribution or a wheel distribution.

exception `pip_accel.exceptions.`**`BinaryDistributionError`**(*text*, ***kw*)
Base class for exceptions related to the generation of binary distributions.

exception `pip_accel.exceptions.`**`InvalidSourceDistribution`**(*text*, ***kw*)
Custom exception raised when a source distribution's setup script is missing.

Raised by `build_binary_dist()` when the given directory doesn't contain a Python source distribution.

exception `pip_accel.exceptions.`**`BuildFailed`**(*text*, ***kw*)
Custom exception raised when a binary distribution build fails.

Raised by `build_binary_dist()` when a binary distribution build fails.

exception `pip_accel.exceptions.`**`NoBuildOutput`**(*text*, ***kw*)
Custom exception raised when binary distribution builds don't generate an archive.

Raised by `build_binary_dist()` when a binary distribution build fails to produce the expected binary distribution archive.

exception `pip_accel.exceptions.`**`CacheBackendError`**(*text*, ***kw*)
Custom exception raised by cache backends when they fail in a controlled manner.

exception `pip_accel.exceptions.`**`CacheBackendDisabledError`**(*text*, ***kw*)
Custom exception raised by cache backends when they require configuration.

exception `pip_accel.exceptions.`**`SystemDependencyError`**(*text*, ***kw*)
Base class for exceptions related to missing system packages.

exception `pip_accel.exceptions.`**`DependencyInstallationRefused`**(*text*, ***kw*)
Custom exception raised when installation of dependencies is refused.

Raised by `SystemPackageManager` when one or more known to be required system packages are missing and automatic installation of missing dependencies is disabled by the operator.

exception `pip_accel.exceptions.`**`DependencyInstallationFailed`**(*text*, ***kw*)
Custom exception raised when installation of dependencies fails.

Raised by `SystemPackageManager` when the installation of missing system packages fails.

pip_accel.tests

Test suite for the pip accelerator.

I've decided to include the test suite in the online documentation of the pip accelerator and I realize this may be somewhat unconventional... My reason for this is to enforce the same level of code quality (which obviously includes documentation) for the test suite that I require from myself and contributors for the other parts of the pip-accel project (and my other open source projects).

A second and more subtle reason is because of a tendency I've noticed in a lot of my projects: Useful "miscellaneous" functionality is born in test suites and eventually makes its way to the public API of the project in question. By writing documentation up front I'm saving my future self time. That may sound silly, but consider that writing documentation is a lot easier when you *don't* have to do so retroactively.

`pip_accel.tests.`**`setUpModule`**`()`
Initialize verbose logging to the terminal.

`pip_accel.tests.`**`tearDownModule`**`()`
Cleanup any temporary directories created by *`create_temporary_directory()`*.

`pip_accel.tests.`**`delete_read_only`**`(`*action*, *pathname*, *exc_info*`)`
Force removal of read only files on Windows.

Based on http://stackoverflow.com/a/21263493/788200. Needed because of https://ci.appveyor.com/project/xolox/pip-accel/build/1.0.24.

`pip_accel.tests.`**`create_temporary_directory`**`(`***kw*`)`
Create a temporary directory that will be cleaned up when the test suite ends.

Parameters **`kw`** – Any keyword arguments are passed on to `tempfile.mkdtemp()`.

Returns The pathname of a directory created using `tempfile.mkdtemp()` (a string).

class `pip_accel.tests.`**`PipAccelTestCase`**`(`*methodName='runTest'*`)`
Container for the tests in the pip-accel test suite.

`setUp``()`
Reset logging verbosity before each test.

`skipTest``(`*text*, **args*, ***kw*`)`
Enable backwards compatible "marking of tests to skip".

By calling this method from a return statement in the test to be skipped the test can be marked as skipped when possible, without breaking the test suite when unittest.TestCase.skipTest() isn't available.

`initialize_pip_accel``(`*load_environment_variables=False*, ***overrides*`)`
Construct an isolated pip accelerator instance.

The pip-accel instance will not load configuration files but it may load environment variables because that's how FakeS3 is enabled on Travis CI (and in my local tests).

Parameters

- **`load_environment_variables`** – If `True` the pip-accel instance will load environment variables (not the default).
- **`overrides`** – Any keyword arguments are set as properties on the *`Config`* instance (overrides for configuration defaults).

`test_related_archives_logic``()`
Test filename translation logic used by *`pip_accel.req.Requirement.related_archives`*.

The *pip_accel.req.escape_name()* function generates regular expression patterns that match the given requirement name literally while treating dashes and underscores as equivalent. This test ensures that the generated regular expression patterns work as expected.

test_environment_validation()
Test the validation of `sys.prefix` versus `$VIRTUAL_ENV`.

This tests the *validate_environment()* method.

test_config_object_handling()
Test that configuration options can be overridden in the Python API.

test_config_file_handling()
Test error handling during loading of configuration files.

This tests the *load_configuration_file()* method.

test_cleanup_of_broken_links()
Verify that broken symbolic links in the source index are cleaned up.

This tests the *clean_source_index()* method.

test_empty_download_cache()
Verify pip-accel's "keeping pip off the internet" logic using an empty cache.

This test downloads, builds and installs pep8 1.6.2 to verify that pip-accel keeps pip off the internet when intended.

test_package_upgrade()
Test installation of newer versions over older versions.

test_package_downgrade()
Test installation of older versions over newer version (package downgrades).

test_s3_backend()
Verify the successful usage of the S3 cache backend.

This test downloads, builds and installs pep8 1.6.2 several times to verify that the S3 cache backend works. It depends on FakeS3.

This test uses a temporary binary index which it wipes after a successful installation and then it installs the exact same package again to test the code path that gets a cached binary distribution archive from the S3 cache backend.

Warning: This test *abuses* FakeS3 in several ways to simulate the handling of error conditions (it's not pretty but it is effective because it significantly increases the coverage of the S3 cache backend):

1. **First the FakeS3 root directory is made read only** to force an error when uploading to S3. This is to test the automatic fall back to a read only S3 bucket.
2. **Then FakeS3 is terminated** to force a failure in the S3 cache backend. This verifies that pip-accel handles the failure of an "optional" cache backend gracefully.

test_wheel_install()
Test the installation of a package from a wheel distribution.

This test installs Paver 1.2.4 (a random package without dependencies that I noticed is available as a Python 2.x and Python 3.x compatible wheel archive on PyPI).

test_bdist_fallback()
Verify that fall back from `bdist_dumb` to `bdist` action works.

This test verifies that pip-accel properly handles `setup.py` scripts that break `python setup.py bdist_dumb` but support `python setup.py bdist` as a fall back. This issue was originally reported based on `Paver==1.2.3` in issue 37, so that's the package used for this test.

`test_installed_files_tracking`()
Verify that tracking of installed files works correctly.

This tests the *`update_installed_files()`* method.

When pip installs a Python package it also creates a file called `installed-files.txt` that contains the pathnames of the files that were installed. This file enables pip to uninstall Python packages later on. Because pip-accel implements its own package installation it also creates the `installed-files.txt` file, in order to enable the user to uninstall a package with pip even if the package was installed using pip-accel.

`test_setuptools_injection`()
Test that `setup.py` scripts are always evaluated using setuptools.

This test installs `docutils==0.12` as a sample package whose `setup.py` script uses *distutils* instead of *setuptools*. Because pip and pip-accel unconditionally evaluate `setup.py` scripts using *setuptools* instead of *distutils* the resulting installation should have an `*.egg-info` metadata directory instead of a file (which is what this test verifies).

`test_requirement_objects`()
Test the public properties of *`pip_accel.req.Requirement`* objects.

This test confirms (amongst other things) that the logic which distinguishes transitive requirements from non-transitive (direct) requirements works correctly (and keeps working as expected :-).

`test_editable_install`()
Test the installation of editable packages using `pip install --editable`.

This test clones the git repository of the Python package *pep8* and installs the package as an editable package.

We want to import the *pep8* module to confirm that it was properly installed but we can't do that in the process that's running the test suite because it will fail with an import error. Python subprocesses however will import the *pep8* module just fine.

This happens because `easy-install.pth` (used for editable packages) is loaded once during startup of the Python interpreter and never refreshed. There's no public, documented way that I know of to refresh `sys.path` (see issue 402 in the Gunicorn issue tracker for a related discussion).

`test_setup_requires_caching`()
Test that *`pip_accel.SetupRequiresPatch`* works as expected.

This test is a bit convoluted because I haven't been able to find a simpler way to ensure that setup requirements can be re-used from the `.eggs` directory managed by pip-accel. A side effect inside the setup script seems to be required, but the setuptools sandbox forbids writing to files outside the build directory so an external command needs to be used ...

`generate_package`(*name*, *version*, *source_index*, *tracker_script*, *find_links=None*, *setup_requires=[]*)
Helper for *`test_setup_requires_caching()`* to generate temporary Python packages.

`test_time_based_cache_invalidation`()
Test default cache invalidation logic (based on modification times).

When a source distribution archive is changed the cached binary distribution archive is invalidated and rebuilt. This test ensures that the default cache invalidation logic (based on modification times of files) works as expected.

test_checksum_based_cache_invalidation()
Test alternate cache invalidation logic (based on checksums).

When a source distribution archive is changed the cached binary distribution archive is invalidated and rebuilt. This test ensures that the alternate cache invalidation logic (based on SHA1 checksums of files) works as expected.

check_cache_invalidation(***overrides*)
Test cache invalidation with the given option(s).

test_cli_install()
Test the pip-accel command line interface by installing a trivial package.

This test provides some test coverage for the pip-accel command line interface, to make sure the command line interface works on all supported versions of Python.

test_cli_usage_message()
Test the pip-accel command line usage message.

test_cli_as_module()
Make sure `python -m pip_accel ...` works.

test_constraint_file_support()
Test support for constraint files.

With the pip 7.x upgrade support for constraint files was added to pip. Due to the way this was implemented in pip the use of constraint files would break pip-accel as reported in issue 63. The issue was since fixed and this test makes sure constraint files remain supported.

test_empty_requirements_file()
Test handling of empty requirements files.

Old versions of pip-accel would raise an internal exception when an empty requirements file was given. This was reported in issue 47 and it was pointed out that pip reports a warning but exits with return code zero. This test makes sure pip-accel now handles empty requirements files the same way pip does.

test_system_package_dependency_installation()
Test the (automatic) installation of required system packages.

This test installs cffi 0.8.6 to confirm that the system packages required by cffi are automatically installed by pip-accel to make the build of cffi succeed.

> **Warning:** This test forces the removal of the system package `libffi-dev` before it tries to install cffi, because without this nasty hack the test would only install required system packages on the first run, because on later runs the required system packages would already be installed. Because of this very non conventional behavior the test is skipped unless the environment variable `PIP_ACCEL_TEST_AUTO_INSTALL=yes` is set (opt-in).

test_system_package_dependency_failures()
Test that unsupported platforms are handled gracefully in system package dependency management.

pep8_git_repo
The pathname of a git clone of the *pep8* package (`None` if git fails).

`pip_accel.tests.`**wipe_directory**(*pathname*)
Delete and recreate a directory.

Parameters **pathname** – The directory's pathname (a string).

`pip_accel.tests.`**create_source_dist**(*sources*)
Create a source distribution archive from a Python package.

Parameters **sources** – A dictionary containing a `setup.py` script (a string).

Returns The pathname of the generated archive (a string).

`pip_accel.tests.`**`uninstall_through_subprocess`**(*package_name*)

Remove an installed Python package by running `pip` as a subprocess.

Parameters **package_name** – The name of the package (a string).

This function is specifically for use in the pip-accel test suite to reliably uninstall a Python package installed in the current environment while avoiding issues caused by stale data in pip and the packages it uses internally. Doesn't complain if the package isn't installed to begin with.

`pip_accel.tests.`**`find_installed_version`**(*package_name*, *encoding='UTF-8'*)

Find the version of an installed package (in a subprocess).

Parameters **package_name** – The name of the package (a string).

Returns The package's version (a string) or `None` if the package can't be found.

This function is specifically for use in the pip-accel test suite to reliably determine the installed version of a Python package in the current environment while avoiding issues caused by stale data in pip and the packages it uses internally.

`pip_accel.tests.`**`find_one_file`**(*directory*, *pattern*)

Use *`find_files()`* to find a file and make sure a single file is matched.

Parameters

- **directory** – The pathname of the directory to be searched (a string).
- **pattern** – The filename pattern to match (a string).

Returns The matched pathname (a string).

Raises `AssertionError` when no file or more than one file is matched.

`pip_accel.tests.`**`find_files`**(*directory*, *pattern*)

Find files whose pathname contains the given substring.

Parameters

- **directory** – The pathname of the directory to be searched (a string).
- **pattern** – The filename pattern to match (a string).

Returns A generator of pathnames (strings).

`pip_accel.tests.`**`try_program`**(*program_name*)

Test that a Python program (installed in the current environment) runs successfully.

This assumes that the program supports the `--help` option, because the program is executed with the `--help` argument to verify that the program runs (`--help` was chose because it implies a lack of side effects).

Parameters **program_name** – The base name of the program to test (a string). The absolute pathname will be calculated by combining `sys.prefix` and this argument.

Raises `AssertionError` when a test fails.

`pip_accel.tests.`**`find_python_program`**(*program_name*)

Get the absolute pathname of a Python program installed in the current environment.

Parameters **name** – The base name of the program (a string).

Returns The absolute pathname of the program (a string).

`pip_accel.tests.`**`generate_nonexisting_pathname`**`()`

Generate a pathname that is expected not to exist.

Returns A pathname (string) that doesn't refer to an existing directory or file on the file system (assuming `random.random()` does what it's documented to do :-).

`pip_accel.tests.`**`test_cli`**`(`**arguments*`)`

Test the pip-accel command line interface.

Runs pip-accel's command line interface inside the current Python process by temporarily changing `sys.argv`, invoking the `pip_accel.cli.main()` function and catching `SystemExit`.

Parameters **`arguments`** – The value that `sys.argv` should be set to (a list of strings).

Returns The exit code of `pip-accel`.

class `pip_accel.tests.`**`CaptureOutput`**

Context manager that captures what's written to `sys.stdout`.

`__init__``()`

Initialize a string IO object to be used as `sys.stdout`.

`__enter__``()`

Start capturing what's written to `sys.stdout`.

`__exit__``(`*exc_type=None*, *exc_value=None*, *traceback=None*`)`

Stop capturing what's written to `sys.stdout`.

`__str__``()`

Get the text written to `sys.stdout`.

class `pip_accel.tests.`**`AptLock`**

Cross-process locking for critical sections to enable parallel execution of the test suite.

`__init__``()`

Initialize an *`AptLock`* object.

class `pip_accel.tests.`**`FakeS3Server`**`(`***options*`)`

Subclass of `ExternalCommand` that manages a temporary FakeS3 server.

`__init__``(`***options*`)`

Initialize a *`FakeS3Server`* object.

`root` **= None**

The pathname of the temporary directory used to store the files required to run the FakeS3 server (a string).

`client_options`

Configuration options for pip-accel to connect with the FakeS3 server.

This is a dictionary of keyword arguments for the *`Config`* initializer to make pip-accel connect with the FakeS3 server.

Python Module Index

Index