versionfinder Documentation

Release 1.1.1

Jason Antman

Sep 18, 2020

Contents

Python package to find the version of another package/distribution, whether installed via pip, setuptools or git

CHAPTER 1

Overview

versionfinder is a library intended to identify the version/source details of a specified Python distribution (usually the one calling it), whether it was installed via pip, setuptools or git. This is intended to allow packages to determine what version they are, beyond what is simply coded in the package:

- For packages installed via pip, return the exact requirement that was installed, even if it was a source control URL (editable or not).
- For packages installed via setuptools, return the installed version.
- For packages that are a git clone, return the URL, commit, tag, and whether the repository is dirty (modified) or not.

This is mainly intended for projects that need to display their version information to users (i.e. for use in filing bug reports or support requests) and wish to be as specific as possible, including whether the package was installed from a fork, a specific tag or commit from a git repo, or has local changes not committed to git.

CHAPTER 2

Requirements

- Python 3.5+

CHAPTER 3

Usage

Versionfinder is primarily intended to return information about the package/ distribution it is called from. As some operations can be quite a bit more time consuming than simply reading a `pkg_resources` or `pip` distribution version, it's recommended that Versionfinder be run once during the startup or initialization of your application/process, and the result stored for later use.

The simplest example is finding the version information for whatever package/distribution contains the calling module. In `mymodule.py`, a module within the "mypackage" package/distribution:

```
import logging
from versionfinder import find_version

# If you are using the python logging module, you'll likely want to
# suppress logging from versionfinder itself, as well as the DEBUG-level
# logging from ``pip`` and ``git``, which are called by versionfinder.
for lname in ['versionfinder', 'pip', 'git']:
    l = logging.getLogger(lname)
    l.setLevel(logging.CRITICAL)
    l.propagate = True

class MyClass(object):

    def __init__(self):
        self._versioninfo = find_version('mypackage')

    @property
    def versioninfo(self):
        return self._versioninfo
```

The `_versioninfo` attribute of the class will be set to the `VersionInfo` object returned by `find_version()`. We can inspect some of that object's properties, which are documented in the API docs.

```
>>> from mypackage.mymodule import MyClass
>>> c = MyClass()
>>> v = c.versioninfo
```

(continues on next page)

(continued from previous page)

```
>>> v
VersionInfo(git_commit=123456ab, git_is_dirty=True, git_remotes={'origin': 'https://
↪github.com/someone/foo.git'}, git_tag=v1.2.3, pip_requirement=git+https://github.
↪com/someone/foo@v1.2.3#egg=foo, pip_url=http://foo.com, pip_version=1.2.3, pkg_
↪resources_url=http://foo.com, pkg_resources_version=1.2.3)
>>> v.pip_version
'1.2.3'
>>> v.pkg_resources_version
'1.2.3'
>>> v.version
'1.2.3'
>>> v.pip_url
'http://foo.com'
>>> v.pkg_resources_url
'http://foo.com'
>>> v.url
'http://foo.com'
>>> v.pip_requirement
'git+https://github.com/someone/foo@v1.2.3#egg=foo'
>>> v.git_remotes
{'origin': 'https://github.com/someone/foo.git'}
>>> v.git_remote
'https://github.com/someone/foo.git'
>>> v.git_commit
'123456ab'
>>> v.git_tag
'v1.2.3'
>>> v.git_is_dirty
True
>>> v.git_str
'git+https://github.com/someone/foo@v1.2.3#egg=foo*'
>>> v.short_str
'1.2.3 <http://foo.com>'
>>> v.long_str
'1.2.3 <http://foo.com> (git+https://github.com/someone/foo@v1.2.3#egg=foo*)'
```

CHAPTER 4

Bugs and Feature Requests

Bug reports and feature requests are happily accepted via the GitHub Issue Tracker. Pull requests are welcome. Issues that don't have an accompanying pull request will be worked on as my time and priority allows.

CHAPTER 5

Development

To install for development:

1. Fork the versionfinder repository on GitHub
2. Create a new branch off of master in your fork.

```
$ virtualenv versionfinder
$ cd versionfinder && source bin/activate
$ pip install -e git+git@github.com:YOURNAME/versionfinder.git@BRANCHNAME
↪#egg=versionfinder
$ cd src/versionfinder
```

The git clone you're now in will probably be checked out to a specific commit, so you may want to `git checkout BRANCHNAME`.

5.1 Guidelines

- pep8 compliant with some exceptions (see pytest.ini)
- 100% test coverage with pytest (with valid tests)

5.2 Testing

Testing is done via pytest, driven by tox.

- testing is as simple as:
  - `pip install tox`
  - `tox`
- If you want to pass additional arguments to pytest, add them to the tox command line after "–". i.e., for verbose pytext output on py27 tests: `tox -e py27 -- -v`

5.3 Acceptance Tests

Versionfinder has a suite of acceptance tests that create virtualenvs, install a test package (versionfinder-test-pkg) in them, and then call `versionfinder.find_version()` from multiple locations in the package, printing a JSON-serialized dict of the results of each call (and an exception, if one was caught). For further information on the acceptance tests, see `versionfinder/tests/test_acceptance.py`.

Currently-tested scenarios are:

- Pip
  - Install from local git clone
  - Install editable from local git clone
  - Install editable from local git clone then change a file (dirty)
  - Install editable from local git clone then commit and tag
  - Install editable from local git clone checked out to a tag
  - Install editable from local git clone checked out to a commit
  - Install editable from local git clone with multiple remotes
  - Install from sdist
  - Install from sdist with pip 1.5.4
  - Install from wheel
  - Install from git URL
  - Install from git fork URL
  - Install from git URL with commit
  - Install from git URL with tag
  - Install from git URL with branch
  - Install editable from git URL
  - Install editable from git fork URL
  - Install editable from git URL with multiple remotes
  - Install editable from git URL and then change a file in the clone (dirty)
  - Install editable from git URL with commit
  - Install editable from git URL with tag
  - Install editable from git URL with branch
  - Install sdist in a venv that's also a git repo
  - Install wheel in a venv that's also a git repo
- setuptools / setup.py
  - setup.py develop
  - setup.py install
- easy_install
  - Install from sdist

  - Install from egg
  - Install from source directory
  - Install from sdist in a venv that's also a git repo
  - Install from egg in a venv that's also a git repo
  - Install from source directory in a venv that's also a git repo

5.4 Release Checklist

1. Open an issue for the release; cut a branch off master for that issue.
2. Confirm that there are CHANGES.rst entries for all major changes.
3. Ensure that Travis tests passing in all environments.
4. Ensure that test coverage is no less than the last release (ideally, 100%).
5. Increment the version number in versionfinder/version.py and add version and release date to CHANGES.rst, then push to GitHub.
6. Confirm that README.rst renders correctly on GitHub.
7. Upload package to testpypi:
   - Make sure your ~/.pypirc file is correct (a repo called `test` for https://testpypi.python.org/pypi)
   - `rm -Rf dist`
   - `python setup.py register -r https://testpypi.python.org/pypi`
   - `python setup.py sdist bdist_wheel`
   - `twine upload -r test dist/*`
   - Check that the README renders at https://testpypi.python.org/pypi/versionfinder
8. Create a pull request for the release to be merged into master. Upon successful Travis build, merge it.
9. Tag the release in Git, push tag to GitHub:
   - tag the release. for now the message is quite simple: `git tag -a X.Y.Z -m 'X.Y.Z released YYYY-MM-DD'`
   - push the tag to GitHub: `git push origin X.Y.Z`
11. Upload package to live pypi:
   - `twine upload dist/*`
10. make sure any GH issues fixed in the release were closed.

5.5 License and Disclaimer

This software is licensed under the GNU Lesser General Public License (LGPL) 3.0.

CHAPTER 6

Contents

6.1 versionfinder

6.1.1 versionfinder package

versionfinder.**find_version**(**args*, ***kwargs*)

Wrapper around *VersionFinder* and its *find_package_version()* method. Pass arguments and kwargs to VersionFinder constructor, return the value of its find_package_version method.

Parameters

- **package_name** (*str*) – name of the package to find information about
- **package_file** (*str*) – absolute path to a Python source file in the package to find information about; if not specified, the file calling this class will be used
- **log** (*bool*) – If not set to True, the "versionfinder" and "pip" loggers will be set to a level of logging.CRITICAL to suppress log output. If set to True, you will see a LOT of debug-level log output, for debugging the internals of versionfinder.

Returns information about the installed version of the package

Return type *VersionInfo*

6.1.1.1 Submodules

versionfinder.version module

versionfinder.versionfinder module

class versionfinder.versionfinder.**VersionFinder**(*package_name*, *package_file=None*, *log=False*, *caller_frame=None*)

Bases: object

_dist_version_url(*dist*)
Get version and homepage for a pkg_resources.Distribution

Parameters **dist** – the pkg_resources.Distribution to get information for

Returns 2-tuple of (version, homepage URL)

Return type tuple

_find_git_info(*gitdir*)
Find information about the git repository, if this file is in a clone.

Parameters **gitdir** (*str*) – path to the git repo's .git directory

Returns information about the git clone

Return type dict

_find_pip_info()
Try to find information about the installed package from pip. This should be wrapped in a try/except.

Returns information from pip about `self.package_name`.

Return type dict

_find_pkg_info()
Find information about the installed package from pkg_resources.

Returns information from pkg_resources about `self.package_name`

Return type dict

_git_repo_path
Attempt to determine whether this package is installed via git or not; if so, return the path to the git repository.

Return type str

Returns path to git repo, or None

_package_top_dir
Find one or more directories that we think may be the top-level directory of the package; return a list of their absolute paths.

Returns list of possible package top-level directories (absolute paths)

Return type list

find_package_version()
Find the installed version of the specified package, and as much information about it as possible (source URL, git ref or tag, etc.)

This attempts, to the best of our ability, to find out if the package was installed from git, and if so, provide information on the origin of that git repository and status of the clone. Otherwise, it uses pip and pkg_resources to find the version and homepage of the installed distribution.

This class is not a sure-fire method of identifying the source of the distribution or ensuring AGPL compliance; it simply helps with this process _iff_ a modified version is installed from an editable git URL _and_ all changes are pushed up to the publicly-visible origin.

Returns a dict with keys 'version', 'tag', 'commit', and 'url'. Values are strings or None.

Parameters **package_name** (*str*) – name of the package to find information for

Returns information about the installed version of the package

Return type *VersionInfo*

versionfinder.versionfinder.**chdir**(**args*, ***kwds*)

versionfinder.versioninfo module

class versionfinder.versioninfo.**VersionInfo**(*pip_version=None*, *pip_url=None*, *pip_requirement=None*, *pkg_resources_version=None*, *pkg_resources_url=None*, *git_tag=None*, *git_commit=None*, *git_remotes=None*, *git_is_dirty=None*)

Bases: object

Class describing *VersionFinder* result; the discovered information about the version and source of an installed package.

as_dict

Return the constructor arguments as a dictionary (effectively the kwargs to the constructor).

Returns dict of constructor arguments

Return type dict

git_commit

Return the hex SHA of the current git commit that the distribution is installed at, or None if not installed via git.

Returns git commit hex SHA

Return type str or None

git_is_dirty

Return True if the distribution is installed via git and has uncommitted changes or untracked files in the repo; Return False if the distribution is installed via git and does not have uncommitted changes or untracked files in the repo; return None if the distribution is not installed via git.

Returns whether or not the git repo has uncommitted changes or untracked files

Return type bool or None

git_remote

If the distribution is installed via git, return the first URL of the 'origin' remote if one is configured for the repo, or else the first URL of the lexicographically-first remote, or else None.

Returns origin or first remote URL

Return type str or None

git_remotes

If the distribution is installed via git, return a dict of all remotes configured on the git repository; keys are the remote name and values are the remote's first URL. If not installed via git, return None.

Returns dict of git remotes, name (str) to first URL (str)

Return type dict or None

git_str

If the distribution is not installed via git, return an empty string.

If the distribution is installed via git and pip recognizes the git source, return the pip requirement string specifying the git URL and commit, with an '*' appended if git_is_dirty() is True.

Otherwise, return a string of the form:

url@ref[*]

Where URL is the remote URL, ref is the tag name if the repo is checked out to a commit that matches a tag or else the commit hex SHA, and '*' is appended if `git_is_dirty()` is True.

Returns description of the git repo remote and state

Return type str

git_tag

Return the name of the git that that the distribution is installed at, or None if there is no tag matching the current commit, or if not installed via git.

Returns current git tag

Return type `str` or `None`

long_str

Return a long version and installation specifier string of the form:

If `git_str()` == '':

SHORT_STR

otherwise:

SHORT_STR (GIT_STR)

Where `SHORT_STR` is `short_str()` and `GIT_STR` is `git_str()`.

Returns long version/installation specifier string

Return type str

pip_requirement

Return the pip requirement for the current installation of the distribution, or None if the distribution cannot be found with pip.

Returns pip requirement string

Return type `str` or `None`

pip_url

Return the pip distribution "Home-page", or None if the distribution cannot be found with pip.

Returns pip distribution Home-page/URL

Return type `str` or `None`

pip_version

Return the pip distribution version, or None if the distribution cannot be found with pip.

Returns pip distribution version

Return type `str` or `None`

pkg_resources_url

Return the pkg_resources distribution "Home-page", or None if the distribution cannot be found with pkg_resources.

Returns pkg_resources distribution Home-page/URL

Return type `str` or `None`

pkg_resources_version

Return the pkg_resources distribution version, or None if the distribution cannot be found with pkg_resources.

Returns pkg_resources distribution version

Return type `str` or `None`

short_str
Return a string of the form "ver <url>" where ver is the distribution version and URL is the distribution Home-Page url, or '' if neither can be found.

Returns version and URL

Return type str

url
Return the package/distribution "Home-page", from pip if possible or else from pkg_resources, or else None if neither can be found.

Returns package/distribution Home-page/URL

Return type `str` or `None`

version
Return the package/distribution version, from pip if possible or else from pkg_resources, or else None if neither can be found.

Returns package/distribution version

Return type `str` or `None`

6.2 Changelog

6.2.1 1.1.1 (2020-09-18)

- Unless *VersionFinder* is constructed with the `log=True` option, completely disable the `pip.subprocessor` logger. This will suppress annoying critical-level log messages generated on systems which do not have `git` in the PATH.

6.2.2 1.1.0 (2020-09-18)

- Switch GitPython requirement from `>=2.1.0,<2.2.0` to `~=3.1`.
- Correct docs to clarify that this package now needs Python >= 3.5.
- Numerous testing changes:
  - Switch tests from deprecated pep8 / pytest-pep8 to pycodestyle / pytest-pycodestyle.
  - Code style fixes for using pycodestyle
  - Remove py27 and py34 test support
  - Update acceptance tests for pip 20

6.2.3 1.0.0 (2019-10-27)

Important: in keeping with the scheduled end-of-life of various Python versions, versionfinder now only officially supports Python 3.5 or greater. A DeprecationWarning will be generated when run with versions before 3.5, and they are no longer tested.

- Fix Issue #7 where certain new versions of pip throw an AttributeError on import if running in Lambda (or other environments where `sys.stdin` is `None`).
- Stop testing Python 3.3 and drop official support for it.
- Stop testing Python 2.7 and 3.4.
- Add DeprecationWarnings for any Python version < 3.5.
- Multiple pip10 fixes.
- Test fixes:
  - Always install latest versions of `coverage` and `pytest`.
  - Switch docs build to py37
  - Begin testing under py37 and py38

6.2.4 0.1.3 (2018-03-18)

- Fix minor unhandled exception in previous release.

6.2.5 0.1.2 (2018-03-18)

- Fix Issue #5 where `import pip` fails if `requests` has previously been imported. Also proactive fix for pip10 changes.
- Multiple test fixes

6.2.6 0.1.1 (2017-06-16)

- Prevent dieing with an exception if `git` is not installed on the system.
- Add hack to `docs/source/conf.py` as workaround for https://github.com/sphinx-doc/sphinx/issues/3860
- Add TravisCI testing for py36

6.2.7 0.1.0 (2016-12-04)

- Initial Release

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index

Symbols

A

C

F

G

L

P

S

U

V