vcversioner Documentation

Release 0.13.1.0

Aaron Gallagher

November 07, 2013

Contents

The code is available on github: https://github.com/habnabit/vcversioner

Elevator pitch: you can write a `setup.py` with no version information specified, and vcversioner will find a recent, properly-formatted git tag and extract a version from it.

It's much more convenient to be able to use your version control system's tagging mechanism to derive a version number than to have to duplicate that information all over the place. I eventually ended up copy-pasting the same code into a couple different `setup.py` files just to avoid duplicating version information. But, copy-pasting is dumb and unit testing `setup.py` files is hard. This code got factored out into vcversioner.

CHAPTER 1

Basic usage

vcversioner installs itself as a setuptools hook, which makes its use exceedingly simple:

```
from setuptools import setup

setup(
    # [...]
    setup_requires=['vcversioner'],
    vcversioner={},
)
```

The presence of a `vcversioner` argument automagically activates vcversioner and updates the project's version. The parameter to the `vcversioner` argument can also be a dict of keyword arguments which `find_version()` will be called with.

To allow tarballs to be distributed without requiring a `.git` directory, vcversioner will also write out a file named (by default) `version.txt`. Then, if there is no git or git is unable to find any version information, vcversioner will read version information from the `version.txt` file. However, this file needs to be included in a distributed tarball, so the following line should be added to `MANIFEST.in`:

```
include version.txt
```

This isn't necessary if `setup.py` will always be run from a git checkout, but otherwise is essential for vcversioner to know what version to use.

The name `version.txt` also can be changed by specifying the `version_file` parameter. For example:

```
from setuptools import setup

setup(
    # [...]
    setup_requires=['vcversioner'],
    vcversioner={
        'version_file': 'custom_version.txt',
    },
)
```

CHAPTER 2

Non-hook usage

It's not necessary to depend on vcversioner; while pip will take care of dependencies automatically, sometimes having a self-contained project is simpler. vcversioner is a single file which is easy to add to a project. Simply copy the entire `vcversioner.py` file adjacent to the existing `setup.py` file and update the usage slightly:

```
from setuptools import setup
import vcversioner

setup(
    # [...]
    version=vcversioner.find_version().version,
)
```

This is necessary because the `vcversioner` distutils hook won't be available.

CHAPTER 3

Version modules

`setup.py` isn't the only place that version information gets duplicated. By generating a version module, the `__init__.py` file of a package can import version information. For example, with a package named `spam`:

```
from setuptools import setup

setup(
    # [...]
    setup_requires=['vcversioner'],
    vcversioner={
        'version_module_paths': ['spam/_version.py'],
    },
)
```

This will generate a `spam/_version.py` file that defines `__version__` and `__sha__`. Then, in `spam/__init__.py`:

```
from spam._version import __version__, __sha__
```

Since this acts like (and *is*) a regular python module, changing `MANIFEST.in` is not required.

CHAPTER 4

Customizing git commands

vcversioner by default executes `git describe --tags --long` to get version information. This command will output a string that describes the current commit, using all tags (as opposed to just unannotated tags), and always output the long format (`1.0-0-gdeadbeef` instead of just `1.0` if the current commit is tagged).

However, sometimes this isn't sufficient. If someone wanted to only use annotated tags, the git command could be amended like so:

```
from setuptools import setup

setup(
    # [...]
    setup_requires=['vcversioner'],
    vcversioner={
        'git_args': ['git', 'describe', '--long'],
    },
)
```

The `git_args` parameter must always be a list of strings, which will not be interpreted by the shell. This is the same as what `subprocess.Popen` expects.

CHAPTER 5

Development versions

vcversioner can also automatically make a version that corresponds to a commit that isn't itself tagged. Following PEP 386, this is done by adding a `.dev` suffix to the version specified by a tag on an earlier commit. For example, if the current commit is three revisions past the `1.0` tag, the computed version will be `1.0.dev3`.

This behavior can be disabled by setting the `include_dev_version` parameter to `False`. In that case, the aforementioned untagged commit's version would be just `1.0`.

CHAPTER 6

Project roots

In order to prevent contamination from other git repositories, vcversioner in the 1.x version series will only look in the project root directory for a git repository. The project root defaults to the current working directory, which is often the case when running setup.py. This can be changed by specifying the `root` parameter. Someone concerned with being able to run setup.py from directories other than the directory containing setup.py should determine the project root from `__file__` in setup.py:

```
from setuptools import setup
import os

setup(
    # [...]
    setup_requires=['vcversioner'],
    vcversioner={
        'root': os.path.dirname(os.path.abspath(__file__)),
    },
)
```

To get the same behavior in the 0.x version series, `git_args` can be set to include the `--git-dir` flag:

```
from setuptools import setup

setup(
    # [...]
    setup_requires=['vcversioner'],
    vcversioner={
        git_args=['git', '--git-dir', '%(root)s/.git', 'describe',
                  '--tags', '--long'],
    },
)
```

By default, `version.txt` is also read from the project root.

CHAPTER 7

Sphinx documentation

Sphinx documentation is yet another place where version numbers get duplicated. Fortunately, since sphinx configuration is python code, vcversioner can be used there too. Assuming vcversioner is installed system-wide, this is quite easy. Since Sphinx is typically run with the current working directory as `<your project root>/docs`, it's necessary to tell vcversioner where the project root is. Simply change your `conf.py` to include:

```
import vcversioner
version = release = vcversioner.find_version(root='..').version
```

This assumes that your project root is the parent directory of the current working directory. A slightly longer version which is a little more robust would be:

```
import vcversioner, os
version = release = vcversioner.find_version(
    root=os.path.dirname(os.path.dirname(os.path.abspath(__file__)))).version
```

This version is more robust because it finds the project root not relative to the current working directory but instead relative to the `conf.py` file.

If vcversioner is bundled with your project instead of relying on it being installed, you might have to add the following to your `conf.py` before `import vcversioner`:

```
import sys, os
sys.path.insert(0, os.path.abspath('..'))
```

This line, or something with the same effect, is sometimes already present when using the sphinx `autodoc` extension.

7.1 Read the Docs

Using vcversioner is even possible when building documentation on Read the Docs. If vcversioner is bundled with your project, nothing further needs to be done. Otherwise, you need to tell Read the Docs to install vcversioner before it builds the documentation. This means using a `requirements.txt` file.

If your project is already set up to install dependencies with a `requirements.txt` file, add `vcversioner` to it. Otherwise, create a `requirements.txt` file. Assuming your documentation is in a `docs` subdirectory of the main project directory, create `docs/requirements.txt` containing a `vcversioner` line.

Then, make the following changes to your project's configuration: (Project configuration is edited at e.g. https://readthedocs.org/dashboard/vcversioner/edit/)

- Check the checkbox under **Use virtualenv**.
- If there was no `requirements.txt` previously, set the **Requirements file** to the newly-created one, e.g. `docs/requirements.txt`.

CHAPTER 8

vcversioner API reference

Simplify your python project versioning.

In-depth docs online: https://vcversioner.readthedocs.org/en/latest/ Code online: https://github.com/habnabit/vcversioner

vcversioner.**find_version**(*include_dev_version=True*, *root=u'%(pwd)s'*, *version_file=u'%(root)s/version.txt'*, *version_module_paths=()*, *git_args=(u'git'*, *u'describe'*, *u'–tags'*, *u'–long')*, *Popen=<class 'subprocess.Popen'>*)

Find an appropriate version number from version control.

It's much more convenient to be able to use your version control system's tagging mechanism to derive a version number than to have to duplicate that information all over the place. Currently, only git is supported.

The default behavior is to write out a `version.txt` file which contains the git output, for systems where git isn't installed or there is no .git directory present. `version.txt` can (and probably should!) be packaged in release tarballs by way of the `MANIFEST.in` file.

Parameters

- **include_dev_version** – By default, if there are any commits after the most recent tag (as reported by git), that number will be included in the version number as a `.dev` suffix. For example, if the most recent tag is `1.0` and there have been three commits after that tag, the version number will be `1.0.dev3`. This behavior can be disabled by setting this parameter to `False`.
- **root** – The directory of the repository root. The default value is the current working directory, since when running `setup.py`, this is often (but not always) the same as the current working directory. Standard substitutions are performed on this value.
- **version_file** – The name of the file where version information will be saved. Reading and writing version files can be disabled altogether by setting this parameter to `None`. Standard substitutions are performed on this value.
- **version_module_paths** – A list of python modules which will be automatically generated containing `__version__` and `__sha__` attributes. For example, with `package/_version.py` as a version module path, `package/__init__.py` could do `from package._version import __version__, __sha__`.
- **git_args** – The git command to run to get a version. By default, this is `git describe --tags --long`. Specify this as a list of string arguments including `git`, e.g. `['git',`

'describe']. Standard substitutions are performed on each value in the provided list.

- **Popen** – Defaults to `subprocess.Popen`. This is for testing.

root, *version_file*, and *git_args* each support some substitutions:

%(root)s The value provided for *root*. This is not available for the *root* parameter itself.

%(pwd)s The current working directory.

vcversioner.**setup**(*dist*, *attr*, *value*)

A hook for simplifying `vcversioner` use from distutils.

This hook, when installed properly, allows vcversioner to automatically run when specifying a `vcversioner` argument to `setup`. For example:

```
from setuptools import setup

setup(
    setup_requires=['vcversioner'],
    vcversioner={},
)
```

The parameter to the `vcversioner` argument is a dict of keyword arguments which `find_version()` will be called with.

Python Module Index

V

vcversioner, ??