vcversioner Documentation

Release 2.16.0.0

Aaron Gallagher

April 12, 2016

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 VCS 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` (or `.hg`, etc.) directory, vcversioner will also write out a file named (by default) `version.txt`. Then, if there is no VCS program or the program 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 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',
    },
)
```

For compatibility with semantic versioning, `vcversioner` will strip leading `'v'`s from version tags. That is, the tag `v1.0` will be treated as if it was `1.0`.

Other prefixes can be specified to be stripped by using the `strip_prefix` argument to vcversioner. For compatibility with `git-dch`, one could specify the `strip_prefix` as `'debian/'`.

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 `__revision__`. Then, in `spam/__init__.py`:

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

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

CHAPTER 4

Customizing VCS commands

vcversioner by default tries to detect which VCS is being used and picks a command to run based on that. For git, that is `git --git-dir %(root)s/.git describe --tags --long`. For hg, that is `hg log -R %(root)s -r . --template '{latesttag}-{latesttagdistance}-hg{node|short}'`.

Any command should output a string that describes the current commit in the format `1.0-0-gdeadbeef`. Specifically, that is `<version number>-<number of commits between the current commit and the version tagged commit>-<revision>`. The revision should have a VCS-specific prefix, e.g. `g` for git and `hg` for hg.

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={
        'vcs_args': ['git', 'describe', '--long'],
    },
)
```

The `vcs_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.

This argument used to be spelled `git_args` until support for multiple VCS systems was added.

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 `.post` 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.post3`.

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`.

Since hg requires a commit to make a tag, there's a parameter `decrement_dev_version` to subtract one from the number of commits after the most recent tag. If the VCS used is detected to be hg (i.e. the revision starts with `'hg'`) and `decrement_dev_version` is not specified as `False`, `decrement_dev_version` will be automatically set to `True`.

CHAPTER 6

Project roots

In order to prevent contamination from other source repositories, vcversioner in the 1.x version series will only look in the project root directory for repositories. 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, `vcs_args` can be set to include the `--git-dir` flag:

```
from setuptools import setup

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

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

6.1 Substitutions

As seen above, *root*, *version_file*, and *vcs_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.

`/` will automatically be translated into the correct path separator for the current platform, such as `:` or `\`.

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=None*, *vcs_args=None*, *decrement_dev_version=None*, *strip_prefix=u'v'*, *Popen=<class 'subprocess.Popen'>*, *open=<built-in function open>*)

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.

The default behavior is to write out a `version.txt` file which contains the VCS output, for systems where the appropriate VCS is not installed or there is no VCS metadata 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 the VCS), that number will be included in the version number as a `.post` 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.post3`. 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** – **Deprecated.** Please use *vcs_args* instead.
- **vcs_args** – The command to run to get a version. By default, this is automatically guessed from directories present in the repository root. Specify this as a list of string arguments

including the program to run, e.g. `['git', 'describe']`. Standard substitutions are performed on each value in the provided list.

- **decrement_dev_version** – If `True`, subtract one from the number of commits after the most recent tag. This is primarily for hg, as hg requires a commit to make a tag. If the VCS used is hg (i.e. the revision starts with `'hg'`) and *decrement_dev_version* is not specified as `False`, *decrement_dev_version* will be set to `True`.
- **strip_prefix** – A string which will be stripped from the start of version number tags. By default this is `'v'`, but could be `'debian/'` for compatibility with `git-dch`.
- **Popen** – Defaults to `subprocess.Popen`. This is for testing.
- **open** – Defaults to `open`. 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.

`/` will automatically be translated into the correct path separator for the current platform, such as `:` or `\`.

`vcversioner` will perform automatic VCS detection with the following directories, in order, and run the specified commands.

`%(root)s/.git`

`git --git-dir %(root)s/.git describe --tags --long. --git-dir` is used to prevent contamination from git repositories which aren't the git repository of your project.

`%(root)s/.hg`

`hg log -R %(root)s -r .  --template '{latesttag}-{latesttagdistance}-hg{node|sho`
`-R` is similarly used to prevent contamination.

`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.

V

F

S

V