
Bump'R Documentation

Release 0.2.1

Axel Haustant

November 21, 2015

1	Compatibility	3
2	Installation	5
3	Usage	7
4	Documentation	9
4.1	Workflow	9
4.2	Configuration file	10
4.3	Command line usage	12
4.4	Hooks	13
4.5	Changelog	15
5	Indices and tables	17

Bump'R is a version bumper and releaser allowing in a single command:

- Clean-up release artifact
- Bump version and tag it
- Build a source distribution and upload on PyPI
- Update version for a new development cycle

Bump'R intend to be customizable with the following features:

- Optionnal test suite run before bump
- Customizable with a config file
- Overridable by command line
- Extensible with hooks

The main goal is to provide a release workflow without manual intervention.

Compatibility

Bump'R requires Python 2.6+

Installation

You can install Bump'R with pip:

```
$ pip install bumpr
```

or with easy_install:

```
$ easy_install bumpr
```

Usage

You can use directly the command line to setup every parameter:

```
$ bumpr fake/__init__.py README.rst -M -ps dev
```

But Bump'R is designed to work with a configuration file (`bumpr.rc` by defaults). Some features are only availables with the configuration file like:

- commit message customization
- hooks configuration
- multiline test, clean and publish commands

Here's an exemple:

```
[bumpr]
file = fake/__init__.py
vcs = git
tests = tox
publish = python setup.py sdist register upload
clean =
    python setup.py clean
    rm -rf *egg-info build dist
files = README.rst

[bump]
unsuffix = true
message = Bump version {version}

[prepare]
suffix = dev
message = Prepare version {version} for next development cycle

[changelog]
file = CHANGELOG.rst
bump = {version} ({date:%Y-%m-%d})
prepare = In development

[readthedoc]
id = fake
```

This way you only have to specify which part you want to bump on the command line:

```
$ bumpr -M -s rc4 # Bump the major with an 'rc4' suffix
$ bumpr          # Bump the default part aka. patch
```


4.1 Workflow

When you execute Bump'R it will follow the following workflow:

1. `clean`
2. `test`
3. `bump`
4. `publish`
5. `prepare`

If you have been using Maven, it's inspired by the Maven Release Plugin.

4.1.1 Clean phase

Optionnal phase that simply execute the commands provided by the `clean` configuration parameter.

4.1.2 Test phase

Optionnal phase that simply execute the commands provided by the `tests` configuration parameter.

4.1.3 Bump phase

This is the main phase in which Bump'R will:

1. Compute replacements
2. Execute the bump phase for each hook
3. Bump replacement in version file and extra files
4. Commit the changes if a VCS is configured with `commit=True`
5. Tag the previously created commit if `tag=True`

4.1.4 Publish phase

Optionnal phase that simply execute the commands provided by the `publish` configuration parameter.

Most of the time for Python project, you will want to execute:

```
python setup.py sdist register upload
```

4.1.5 Prepare phase

This is the second main phase in which Bump'R will:

1. Compute replacements
2. Execute the prepare phase for each hook
3. Bump replacement in version file and extra files
4. Commit the changes if a VCS is configured with `commit=True`

4.2 Configuration file

The `bump.r.c` configuration file is an inifile with the following possible sections and keys.

4.2.1 bump.r

This is the main section defining the common behavior and parameters.

file *Default:* None

The file containing the version string to extract.

regex *Default:* `r'(__version__|VERSION)\s*=\s*(\'|")(?P<version>.+?)(\'|")'`

The regex used to extract the version string. It must have a `version` named group.

encoding *Default:* `utf8`

The files encoding.

vcs *Default:* None

commit *Default:* `True`

If `True` and `vcs` is defined, commit the changes.

tag *Default:* `True`

If `True` and `vcs` is defined, tag the version.

verbose *Default:* `False`

If `True`, display verbose output and command line output.

dryrun *Default:* `False`

If `True`, no command or VCS operation will be executed. They will be displayed in the command output.

clean *Default:* None

Specify the commands to be executed on the `clean` phase. Should have a single command by line.

tests *Default:* None

Specify the commands to be executed on the *test* phase. Should have a single command by line.

publish *Default:* None

Specify the commands to be executed on the *publish* phase. Should have a single command by line.

files *Default:* []

Extra files to process. Those files will be processed by hooks to. Specify one file by line.

4.2.2 bump

This section define the bump phase behavior.

unsuffix *Default:* True

If True the current verion suffix will be removed.

suffix: *Default:* None

If set, this suffix will ba appended to the version.

part: *Default:* None

Specify the part to bump between major, minor or patch.

message *Default:* Bump version {version}

Specify the commit message that will be bumped. You can use the following token in your format pattern: version, major, minor, patch and date. All formating operations are accepted.

4.2.3 prepare

This section define the prepare phase behavior.

unsuffix *Default:* False

If True the current verion suffix will be removed.

suffix: *Default:* None

If set, this suffix will ba appended to the version.

part: *Default:* patch

Specify the part to bump between major, minor or patch.

message *Default:* Update to version {version} for next development cycle

Specify the commit message that will be bumped. You can use the following token in your format pattern: version, major, minor, patch and date. All formating operations are accepted.

4.2.4 hooks

Each hook can contribute to configuration with its own section.

See [Hooks](#).

4.2.5 sample

Here a sample `bumpr.rc` file

```
[bumpr]
file = fake/__init__.py
vcs = git
tests = tox
publish = python setup.py register sdist upload
clean =
    python setup.py clean
    rm -rf *egg-info build dist
files = README.rst

[bump]
message = 'Commit version {version}'

[prepare]
suffix = dev
message = Prepare version {version} for next development cycle

[changelog]
file = CHANGELOG.rst
bump = {version} ({date:%Y-%m-%d})
prepare = In development

[readthedoc]
id = bumpr
```

4.3 Command line usage

The `bumpr` executable can be used in two way:

- one shot usage with all parameters on command line
- regular usage with most of the parameters in a configuration file

The default configuration file name is `bumpr.rc` but you can override it with the `-c` option.

All mandatory parameters not present in the configuration file should be on command line.

```
$ bumpr -h
usage: bumpr [-h] [--version] [-v] [-c CONFIG] [-d] [-b | -pr] [-M] [-m] [-p]
             [-s SUFFIX] [-u] [-pM] [-pm] [-pp] [-ps PREPARE_SUFFIX] [-pu]
             [--vcs {git,hg}] [-nc]
             [file] [files [files ...]]

Version bumper and Python package releaser

positional arguments:
  file                  Versionned module file
  files                 Files to update

optional arguments:
  -h, --help            show this help message and exit
  --version             show program's version number and exit
  -v, --verbose         Verbose output
  -c CONFIG, --config CONFIG
```



```

Specify a configuration file
-d, --dryrun      Do not write anything and display a diff
-b, --bump        Only perform the bump
-pr, --prepare    Only perform the prepare

bump:
-M, --major       Bump major version
-m, --minor       Bump minor version
-p, --patch       Bump patch version
-s SUFFIX, --suffix SUFFIX
                  Set suffix
-u, --unsuffix    Unset suffix

prepare:
-pM, --prepare-major Bump major version
-pm, --prepare-minor Bump minor version
-pp, --prepare-patch Bump patch version
-ps PREPARE_SUFFIX, --prepare-suffix PREPARE_SUFFIX
                  Set suffix
-pu, --prepare-unsuffix
                  Unset suffix

Version control system:
--vcs {git,hg}    VCS implementation
-nc, --nocommit   Do not commit

```

4.4 Hooks

4.4.1 Read the doc

This hook allow to change the Read The Doc documentation URL in both bump and prepare phase.

- **key:** readthedoc
- **parameters**
 - id: None
 - url: `http://{id}.readthedocs.org/en/{tag}`
 - bump: {version}
 - prepare: latest

Most of the the time, you will just have to specify your readthedoc project identifier.

```
[readthedoc]
id = myproject
```

You can customize the generated (and parsed) URL and the string to bump in.

```
[readthedoc]
url = http://custom.doc/{tag}
```

You can also customize the tag part in the url for the bump and prepare phase

```
[readthedoc]
bump = {version}
prepare = latest
```

4.4.2 Changelog

This hook allow to bump and prepare your changelog. In the bump phase it will bump the current developpement header with a versionned one. In the prepare phase, it will do the inverse operation.

- **key:** changelog
- **parameters**
 - file: None
 - separator: -
 - bump: {version} ({date:%Y-%m-%d})'
 - prepare: Current
 - empty: Nothing yet

The file parameter is mandatory and designate the changelog file.

The separator parameter spcify the character user to underline your changelog section.

```
Changelog
=====

In development
~~~~~

- Another feature

Version 1.0.1 (2013-08-23)
~~~~~

- Some new feature
```

To handle this changelog you will have the following configuration

```
[changelog]
file = CHANGELOG
separator = ~
bump = Version {version} ({date:%Y-%m-%d})
prepare = In development
empty = Empty
```

If you execute Bump'R, the changelog will be bumped like:

```
Changelog
=====

Version 1.0.2 (2013-08-24)
~~~~~

- Another feature

Version 1.0.1 (2013-08-23)
~~~~~

- Some new feature
```

And then prepared:

```

Changelog
=====

In development
~~~~~

- Empty

Version 1.0.2 (2013-08-24)
~~~~~

- Another feature

Version 1.0.1 (2013-08-23)
~~~~~

- Some new feature

```

4.4.3 Commands

This hook allow to execute custom commands during bump and prepare phases.

- **key:** `commands`
- **parameters**
 - `bump: {version}`
 - `prepare: latest`

Both bump and prepare command can be multiline (a command statement by line), and support the following format token:

- **version:** the current phase version string
- **major:** the current phase version major part
- **minor:** the current phase version minor part
- **patch:** the current phase version patch part
- **date:** the current date (aka. the release date)

In the bump phase, version will be the bumped version whereas in the prepare phase it will be the prepared/next version.

Exemple:

```

[commands]
bump = echo "{major}.{minor} - {date:%Y-%m-%d} "
prepare = echo "Next version: {version}"

```

4.5 Changelog

4.5.1 0.2.1 (2015-11-21)

- Use nosetests instead of custom discovery
- Some fixes on Python 3 (mostly encodings)

- Improve error handling
- Validate configuration

4.5.2 0.2.0 (2013-08-24)

- colored diff
- Added `--bump` and `--prepare` to only perform bump or prepare
- Rely on VCS for tracking files and ensure working copy is clean
- Added option `--nocommit`
- Ensure dry run does not write or execute anything
- Better output and error handling
- Group parameters in help
- Added optionnal hook validation
- Fix some Python incompatibilities (Python 2.6 and 3.X)
- More documentation

4.5.3 0.1.0 (2013-08-22)

- Initial release. Missing some parts but working!

Indices and tables

- `genindex`
- `modindex`
- `search`