smap

Release 0.2.5

Jul 26, 2018

Contents

CHAPTER 1

smap

A helper for library maintainers to use symbol versioning

1.1 Why use symbol versioning?

The main reason is to be able to keep the library *[ABI]* stable.

If a library is intended to be used for a long time, it will need updates for eventual bug fixes and/or improvement. This can lead to changes in the *[API]* and, in the worst case, changes to the *[ABI]*.

Using symbol versioning, it is possible to make compatible changes and keep the applications working without recompiling. If incompatible changes were made (breaking the *[ABI]*), symbol versioning allows both incompatible versions to live in the same system without conflict. And even more uncommon situations, like an application to be linked to different (incompatible) versions of the same library.

For more information, I strongly recommend reading:

- *[HOW_TO]* How to write shared libraries, by Ulrich Drepper

1.2 How to add symbol versioning to my library?

Adding version information to the symbols is easy. Keeping the *[ABI]* stable, unfortunately, is not. This project intends to help in the first part.

To add version information to symbols of a library, one can use version scripts (in Linux). Version scripts are files used by linkers to map symbols to a given version. It contains the symbols exported by the library grouped by the releases where they were introduced. For example:

```
LIB_EXAMPLE_1_0_0
  {
    global:
      symbol;
```

(continues on next page)

(continued from previous page)

```
      another_symbol;
    local:
      *;
  };
```

In this example, the release `LIB_EXAMPLE_1_0_0` introduces the symbols `symbol` and `another_symbol`. The `*` wildcard in `local` catches all other symbols, meaning only `symbol` and `another_symbol` are globally exported as part of the library *[API]*.

If a compatible change is made, it would introduce a new release, like:

```
LIB_EXAMPLE_1_1_0
{
    global:
        new_symbol;
} LIB_EXAMPLE_1_0_0;

LIB_EXAMPLE_1_0_0
{
    global:
        symbol;
        another_symbol;
    local:
        *;
};
```

The new release `LIB_EXAMPLE_1_1_0` introduces the symbol `new_symbol`. The `*` wildcard should be only in one version, usually in the oldest version. The `} LIB_EXAMPLE_1_0_0;` part in the end of the new release means the new release depends on the old release.

Suppose a new incompatible version `LIB_EXAMPLE_2_0_0` released after `LIB_EXAMPLE_1_1_0`. Its map would look like:

```
LIB_EXAMPLE_2_0_0
{
    global:
        a_newer_symbol;
        another_symbol;
        new_symbol;
    local:
        *;
};
```

The symbol `symbol` was removed (and that is why it was incompatible). And a new symbol was introduced, `a_newer_symbol`.

Note that all global symbols in all releases were merged in a unique new release.

1.3 Installation:

At the command line:

```
pip install symver-smap
```

1.4 Usage:

This project delivers a script, `smap`. This is my first project in python, so feel free to point out ways to improve it.

The sub-commands `update` and `new` expect a list of symbols given in stdin. The list of symbols are words separated by non-alphanumeric characters (matches with the regular expression `[a-zA-Z0-9_]+`). For example:

```
symbol, another, one_more
```

and:

```
symbol
another
one_more
```

are valid inputs.

The last sub-command, `check`, expects only the path to the map file to be checked.

1.5 tl;dr

```
$ smap update lib_example.map < symbols_list
```

or (setting an output):

```
$ smap update lib_example.map -o new.map < symbols_list
```

or:

```
$ cat symbols_list | smap update lib_example.map -o new.map
```

or (to create a new map):

```
$ cat symbols_list | smap new -r lib_example_1_0_0 -o new.map
```

or (to check the content of a existing map):

```
$ smap check my.map
```

or (to check the current version):

```
$ smap version
```

1.6 Long version

Running `smap -h` will give:

```
usage: smap [-h] {update,new,check,version} ...

Helper tools for linker version script maintenance

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help            show this help message and exit

Subcommands:
  {update,new,check,version}
                        These subcommands have their own set of options
    update              Update the map file
    new                 Create a new map file
    check               Check the map file
    version             Print version

Call a subcommand passing '-h' to see its specific options
```

Call a subcommand passing '-h' to see its specific options There are four subcommands, `update`, `new`, `check`, and `version`

Running `smap update -h` will give:

```
usage: smap update [-h] [-o OUT] [-i INPUT] [-d]
                   [--verbosity {quiet,error,warning,info,debug} | --quiet | --debug]
                   [-l LOGFILE] [-n NAME] [-v VERSION] [-r RELEASE]
                   [--no_guess] [--allow-abi-break] [-f] [-a | --remove]
                   file

positional arguments:
  file                  The map file being updated

optional arguments:
  -h, --help            show this help message and exit
  -o OUT, --out OUT     Output file (defaults to stdout)
  -i INPUT, --in INPUT  Read from this file instead of stdio
  -d, --dry             Do everything, but do not modify the files
  --verbosity {quiet,error,warning,info,debug}
                        Set the program verbosity
  --quiet               Makes the program quiet
  --debug               Makes the program print debug info
  -l LOGFILE, --logfile LOGFILE
                        Log to this file
  -n NAME, --name NAME  The name of the library (e.g. libx)
  -v VERSION, --version VERSION
                        The release version (e.g. 1_0_0 or 1.0.0)
  -r RELEASE, --release RELEASE
                        The full name of the release to be used (e.g.
                        LIBX_1_0_0)
  --no_guess            Disable next release name guessing
  --allow-abi-break     Allow removing symbols, and to break ABI
  -f, --final           Mark the modified release as final, preventing later
                        changes.
  -a, --add             Adds the symbols to the map file.
  --remove              Remove the symbols from the map file. This breaks the
                        ABI.

A list of symbols is expected as the input. If a file is provided with '-i',
the symbols are read from the given file. Otherwise the symbols are read from
stdin.
```

Running `smap new -h` will give:

```
usage: smap new [-h] [-o OUT] [-i INPUT] [-d]
                [--verbosity {quiet,error,warning,info,debug} | --quiet | --debug]
                [-l LOGFILE] [-n NAME] [-v VERSION] [-r RELEASE] [--no_guess]
                [-f]

optional arguments:
  -h, --help            show this help message and exit
  -o OUT, --out OUT     Output file (defaults to stdout)
  -i INPUT, --in INPUT  Read from this file instead of stdio
  -d, --dry             Do everything, but do not modify the files
  --verbosity {quiet,error,warning,info,debug}
                        Set the program verbosity
  --quiet               Makes the program quiet
  --debug               Makes the program print debug info
  -l LOGFILE, --logfile LOGFILE
                        Log to this file
  -n NAME, --name NAME  The name of the library (e.g. libx)
  -v VERSION, --version VERSION
                        The release version (e.g. 1_0_0 or 1.0.0)
  -r RELEASE, --release RELEASE
                        The full name of the release to be used (e.g.
                        LIBX_1_0_0)
  --no_guess            Disable next release name guessing
  -f, --final           Mark the new release as final, preventing later
                        changes.

A list of symbols is expected as the input. If a file is provided with '-i',
the symbols are read from the given file. Otherwise the symbols are read from
stdin.
```

Running `smap check -h` will give:

```
usage: smap check [-h]
                  [--verbosity {quiet,error,warning,info,debug} | --quiet | --debug]
                  [-l LOGFILE]
                  file

positional arguments:
  file                  The map file to be checked

optional arguments:
  -h, --help            show this help message and exit
  --verbosity {quiet,error,warning,info,debug}
                        Set the program verbosity
  --quiet               Makes the program quiet
  --debug               Makes the program print debug info
  -l LOGFILE, --logfile LOGFILE
                        Log to this file
```

Running `smap version -h` will give:

```
usage: smap version [-h]

optional arguments:
  -h, --help  show this help message and exit
```

1.7 Import as a library:

To use smap in a project as a library:

```
from smap import symver
```

1.8 Documentation:

Check in Read the docs

1.9 References:

CHAPTER 2

Installation

At the command line:

```
pip install symver-smap
```

CHAPTER 3

Usage

This project delivers a script, `smap`. This is my first project in python, so feel free to point out ways to improve it.

The sub-commands `update` and `new` expect a list of symbols given in stdin. The list of symbols are words separated by non-alphanumeric characters (matches with the regular expression `[a-zA-Z0-9_]+`). For example:

```
symbol, another, one_more
```

and:

```
symbol
another
one_more
```

are valid inputs.

The last sub-command, `check`, expects only the path to the map file to be checked.

3.1 tl;dr

```
$ smap update lib_example.map < symbols_list
```

or (setting an output):

```
$ smap update lib_example.map -o new.map < symbols_list
```

or:

```
$ cat symbols_list | smap update lib_example.map -o new.map
```

or (to create a new map):

```
$ cat symbols_list | smap new -r lib_example_1_0_0 -o new.map
```

or (to check the content of a existing map):

```
$ smap check my.map
```

or (to check the current version):

```
$ smap version
```

3.2 Long version

Running `smap -h` will give:

```
usage: smap [-h] {update,new,check,version} ...

Helper tools for linker version script maintenance

optional arguments:
  -h, --help            show this help message and exit

Subcommands:
  {update,new,check,version}
                        These subcommands have their own set of options
    update              Update the map file
    new                 Create a new map file
    check               Check the map file
    version             Print version

Call a subcommand passing '-h' to see its specific options
```

Call a subcommand passing '-h' to see its specific options There are four subcommands, `update`, `new`, `check`, and `version`

Running `smap update -h` will give:

```
usage: smap update [-h] [-o OUT] [-i INPUT] [-d]
                   [--verbosity {quiet,error,warning,info,debug} | --quiet | --debug]
                   [-l LOGFILE] [-n NAME] [-v VERSION] [-r RELEASE]
                   [--no_guess] [--allow-abi-break] [-f] [-a | --remove]
                   file

positional arguments:
  file                  The map file being updated

optional arguments:
  -h, --help            show this help message and exit
  -o OUT, --out OUT     Output file (defaults to stdout)
  -i INPUT, --in INPUT  Read from this file instead of stdio
  -d, --dry             Do everything, but do not modify the files
  --verbosity {quiet,error,warning,info,debug}
                        Set the program verbosity
  --quiet               Makes the program quiet
  --debug               Makes the program print debug info
  -l LOGFILE, --logfile LOGFILE
                        Log to this file
  -n NAME, --name NAME  The name of the library (e.g. libx)
  -v VERSION, --version VERSION
```

(continues on next page)

(continued from previous page)

```
                        The release version (e.g. 1_0_0 or 1.0.0)
  -r RELEASE, --release RELEASE
                        The full name of the release to be used (e.g.
                        LIBX_1_0_0)
  --no_guess            Disable next release name guessing
  --allow-abi-break     Allow removing symbols, and to break ABI
  -f, --final           Mark the modified release as final, preventing later
                        changes.
  -a, --add             Adds the symbols to the map file.
  --remove              Remove the symbols from the map file. This breaks the
                        ABI.

A list of symbols is expected as the input. If a file is provided with '-i',
the symbols are read from the given file. Otherwise the symbols are read from
stdin.
```

Running `smap new -h` will give:

```
usage: smap new [-h] [-o OUT] [-i INPUT] [-d]
                [--verbosity {quiet,error,warning,info,debug} | --quiet | --debug]
                [-l LOGFILE] [-n NAME] [-v VERSION] [-r RELEASE] [--no_guess]
                [-f]

optional arguments:
  -h, --help            show this help message and exit
  -o OUT, --out OUT     Output file (defaults to stdout)
  -i INPUT, --in INPUT  Read from this file instead of stdio
  -d, --dry             Do everything, but do not modify the files
  --verbosity {quiet,error,warning,info,debug}
                        Set the program verbosity
  --quiet               Makes the program quiet
  --debug               Makes the program print debug info
  -l LOGFILE, --logfile LOGFILE
                        Log to this file
  -n NAME, --name NAME  The name of the library (e.g. libx)
  -v VERSION, --version VERSION
                        The release version (e.g. 1_0_0 or 1.0.0)
  -r RELEASE, --release RELEASE
                        The full name of the release to be used (e.g.
                        LIBX_1_0_0)
  --no_guess            Disable next release name guessing
  -f, --final           Mark the new release as final, preventing later
                        changes.

A list of symbols is expected as the input. If a file is provided with '-i',
the symbols are read from the given file. Otherwise the symbols are read from
stdin.
```

Running `smap check -h` will give:

```
usage: smap check [-h]
                  [--verbosity {quiet,error,warning,info,debug} | --quiet | --debug]
                  [-l LOGFILE]
                  file

positional arguments:
  file                  The map file to be checked
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  --verbosity {quiet,error,warning,info,debug}
                        Set the program verbosity
  --quiet               Makes the program quiet
  --debug               Makes the program print debug info
  -l LOGFILE, --logfile LOGFILE
                        Log to this file
```

Running `smap version -h` will give:

```
usage: smap version [-h]

optional arguments:
  -h, --help  show this help message and exit
```

3.3 Import as a library:

To use smap in a project as a library:

```
from smap import symver
```

CHAPTER 4

Reference

4.1 smap package

4.1.1 Submodules

4.1.2 smap.main module

Entrypoint used to generate the command line application

`smap.main.`**`main`**`()`

4.1.3 smap.symver module

class `smap.symver.`**`Map`**(*filename=None*, *logger=None*)

Bases: `object`

A linker map (version script) representation

This class is an internal representation of a version script. It is intended to be initialized by calling the method `read()` and passing the path to a version script file. The parser will parse the file and check the file syntax, creating a list of releases (instances of the `Release` class), which is stored in `releases`.

Variables

- **`init`** – Indicates if the object was initialized by calling `read()`
- **`logger`** – The logger object; can be specified in the constructor
- **`filename`** – Holds the name (path) of the file read
- **`lines`** – A list containing the lines of the file

`all_global_symbols``()`

Returns all global symbols from all releases contained in the Map object

Returns A set containing all global symbols in all releases

check()
Check the map structure.

Reports errors found in the structure of the map in form of warnings.

dependencies()
Construct the dependencies lists

Contruct a list of dependency lists. Each dependency list contain the names of the releases in a dependency path. The heads of the dependencies lists are the releases not refered as a previous release in any release.

Returns A list containing the dependencies lists

duplicates()
Find and return a list of duplicated symbols for each release

If no duplicates are found, return an empty list

Returns A list of tuples [(release, [(scope, [duplicates])])]

guess_latest_release()
Try to guess the latest release

It uses the information found in the releases present in the version script read. It tries to find the latest release using heuristics.

Returns A list [release, prefix, suffix, version[CUR, AGE, REV]]

guess_name(*new_release*, *abi_break=False*, *guess=False*)
Use the given information to guess the name for the new release

The two parts necessary to make the release name:

- The new prefix: Usually the library name (e.g. LIBX)
- The new suffix: The version information (e.g. _1_2_3)

If the new release is not provided, try a guess strategy:

If the new prefix is not provided:

1. Try to find a common prefix between release names
2. Try to find latest release

If the new suffix is not provided:

1. Try to find latest release version and bump

Parameters

- **new_release** – String, the name of the new release. If this is
- **abi_break** – Boolean, indicates if the ABI was broken
- **guess** – Boolean, indicates if should try to guess

Returns The guessed release name (new prefix + new suffix)

parse(*lines*)
A simple version script parser.

This is the main initializator of the `releases` list. This simple parser receives the lines of a given version script, check its syntax, and construct the list of releases. Some semantic aspects are checked, like the existence of the `*` wildcard in global scope and the existence of duplicated release names.

It works by running a finite state machine:

The parser states. Can be:

0. name: The parser is searching for a release name or `EOF`
1. opening: The parser is searching for the release opening `{`
2. element: The parser is searching for an identifier name or `}`
3. element_closer: The parser is searching for `:` or `;`
4. previous: The parser is searching for previous release name
5. previous_closer: The parser is searching for `;`

Parameters **`lines`** – The lines of a version script file

`read`(*filename*)

Read a linker map file (version script) and store the obtained releases

Obtain the lines of the file and calls `parse()` to parse the file

Parameters **`filename`** – The path to the file to be read

Raises ***`ParserError`*** – Raised when a syntax error is found in the file

`sort_releases_nice`(*top_release*)

Sort the releases contained in a map file putting the dependencies of `top_release` first. This changes the order of the list in `releases`.

Parameters **`top_release`** – The release whose dependencies should be prioritized

`exception` `smap.symver.`**`ParserError`**(*filename*, *context*, *line*, *column*, *message*)

Bases: `exceptions.Exception`

Exception type raised by the map parser

Used mostly to keep track where an error was found in the given file

Variables

- **`filename`** – The name (path) of the file being parsed
- **`context`** – The line where the error was detected
- **`line`** – The index of the line where the error was detected
- **`column`** – The index of the column where the error was detected
- **`message`** – The error message

`class` `smap.symver.`**`Release`**

Bases: `object`

A internal representation of a release version and its symbols

A release is usually identified by the library name (suffix) and the release version (suffix). A release contains symbols, grouped by their visibility scope (global or local).

In this class the symbols of a release are stored in a list of dictionaries mapping a visibility scope name (e.g. “global”) to a list of the contained symbols:

```
([{"global": [symbols]}, {"local": [local_symbols]}])
```

Variables

- **`name`** – The release name

- **previous** – The previous release to which this release is dependent
- **symbols** – The symbols contained in the release, grouped by the visibility scope.

duplicates()

class smap.symver.**Single_Logger**
Bases: object

A singleton logger for the module

This class is a singleton logger factory. It takes advantage of the uniqueness of class attributes to hold a unique instance of the logger for the module. It logs to the default log output, and prints WARNING and ERROR messages to stderr. It allows the caller to provide a file to receive the log (the messages will be logged by all handlers: to stderr if WARNING or ERROR, to default log, and to the provided file)

Variables **__instance** – Holds the unique instance given by the factory when called.

classmethod getLogger(*name*, *filename=None*)
Get the unique instance of the logger

Parameters **name** – The name of the module (usually just __name__)

Returns An instance of logging.Logger

smap.symver.**bump_version**(*version*, *abi_break*)
Bump a version depending if the ABI was broken or not

If the ABI was broken, CUR is bumped; AGE and REV are set to zero. Otherwise, CUR is kept, AGE is bumped, and REV is set to zero. This also works with versions without the REV component (e.g. [1, 4, None])

Parameters

- **version** – A list in format [CUR, AGE, REV]
- **abi_break** – A boolean indication if the ABI was broken

Returns A list in format [CUR, AGE, REV]

smap.symver.**check**(*args*)
'check' subcommand

Check the content of a symbol version script

Parameters **args** – Arguments given in command line parsed by argparse

smap.symver.**check_files**(*out_arg*, *out_name*, *in_arg*, *in_name*, *dry*)
Check if output and input are the same file. Create a backup if so.

Parameters

- **out_arg** – The name of the option used to receive output file name
- **out_name** – The received string as output file path
- **in_arg** – The name of the option used to receive input file name
- **in_name** – The received string as input file path

smap.symver.**clean_symbols**(*symbols*)
Receives a list of lines read from the input and returns a list of words

Parameters **symbols** – A list of lines containing symbols

Returns A list of the obtained symbols

smap.symver.**get_arg_parser**()
Get a parser for the command line arguments

The parser is capable of checking requirements for the arguments and possible incompatible arguments.

Returns A parser for command line arguments. (argparse.ArgumentParser)

smap.symver.**get_info_from_args**(*args*)
Get the release information from the provided arguments

It is possible to set the new release name to be used through the command line arguments.

Parameters **args** – Arguments given in command line parsed by argparse

smap.symver.**get_info_from_release_string**(*release*)
Get the information from a release name

The given string is split in a prefix (usually the name of the lib) and a suffix (the version part, e.g. '_1_4_7'). A list with the version info converted to ints is also contained in the returned list.

Parameters **release** – A string in format 'LIBX_1_0_0' or similar

Returns A list in format [release, prefix, suffix, [CUR, AGE, REV]]

smap.symver.**get_version_from_string**(*version_string*)
Get the version numbers from a string

Parameters **version_string** – A string composed by numbers separated by non alphanumeric characters (e.g. 0_1_2 or 0.1.2)

Returns A list of the numbers in the string

smap.symver.**new**(*args*)
'new' subcommand

Create a new version script file containing the provided symbols.

Parameters **args** – Arguments given in command line parsed by argparse

smap.symver.**update**(*args*)
Given the new list of symbols, update the map

The new map will be generated by the following rules:

- If new symbols are added, a new release is created containing the new symbols. This is a compatible update.
- If a previous existing symbol is removed, then all releases are unified in a new release. This is an incompatible change, the SONAME of the library should be bumped

The symbols provided are considered all the exported symbols in the new version. Such set of symbols is compared to the previous existing symbols. If symbols are added, but nothing removed, it is a compatible change. Otherwise, it is an incompatible change and the SONAME of the library should be bumped.

If –add is provided, the symbols provided are considered new symbols to be added. This is a compatible change.

If –remove is provided, the symbols provided are considered the symbols to be removed. This is an incompatible change and the SONAME of the library should be bumped.

Parameters **args** – Arguments given in command line parsed by argparse

smap.symver.**version**(*args*)
'version' subcommand

Prints and returns the program name and version.

Parameters **args** – Arguments given in command line parsed by argparse

Returns A string containing the program name and version

4.1.4 Module contents

4.2 smap

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/ansasaki/smap/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

smap could always use more documentation, whether as part of the official smap docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ansasaki/smap/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *smap* for local development.

1. Fork the *smap* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/smap.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv smap
$ cd smap/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 smap tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/ansasaki/smap/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_smap
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Anderson Toshiyuki Sasaki <ansasaki@redhat.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

Changelog

7.1 0.2.5 (2018-07-26)

- Add tests using different program names
- Use the command line application name in output strings
- Add a new entry point symver-smap for console scripts
- Skip tests which use caplog if pytest version is < 3.4
- Added an alias for pytest in setup.cfg. This fixed setup.py for test target

7.2 0.2.4 (2018-06-15)

- Removed dead code, removed executable file permission
- Removed appveyor related files

7.3 0.2.3 (2018-06-15)

- Removed shebangs from scripts

7.4 0.2.2 (2018-06-01)

- Fixed a bug in updates with provided release information
- Fixed a bug in get_info_from_release_string()

7.5 0.2.1 (2018-05-30)

- Fixed a bug where invalid characters were accepted in release name

7.6 0.2.0 (2018-05-29)

- Added version information in output files
- Added sub-command "version" to output name and version
- Added option "–final" to mark modified release as released
- Prevent releases marked with the special comment "# Released" to be modified
- Allow existing release update
- Detect duplicated symbols given as input

7.7 0.1.0 (2018-05-09)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Bibliography

[ABI] https://en.wikipedia.org/wiki/Application_binary_interface

[API] https://en.wikipedia.org/wiki/Application_programming_interface

[HOW_TO] https://www.akkadia.org/drepper/dsohowto.pdf, How to write shared libraries by Ulrich Drepper

Python Module Index

S

Index