scriv

Release 1.2.0

Ned Batchelder

Jan 18, 2023

Contents

1 Overview 3

2 Getting Started 5

Scriv changelog management tool

CHAPTER 1

Overview

Scriv is a command-line tool for helping developers maintain useful changelogs. It manages a directory of changelog fragments. It aggregates them into entries in a CHANGELOG file.

Currently scriv implements a simple workflow. The goal is to adapt to more styles of changelog management in the future.

CHAPTER 2

Getting Started

Scriv writes changelog fragments into a directory called "changelog.d". Start by creating this directory. (By the way, like many aspects of scriv's operation, you can choose a different name for this directory.)

To make a new changelog fragment, use the "*scriv create*" command. It will make a new file with a filename using the current date and time, your GitHub or Git user name, and your branch name. Changelog fragments should be committed along with all the other changes on your branch.

When it is time to release your project, the "*scriv collect*" command aggregates all the fragments into a new entry in your changelog file.

You can also choose to publish your changelog entries as GitHub releases with the "*scriv github-release*" command. It parses the changelog file and creates or updates GitHub releases to match. It can be used even with changelog files that were not created by scriv.

2.1 Philosophy

Scriv's design is guided by a few principles:

- Changelogs should be captured in a file in the repository. Scriv writes a CHANGELOG file.
- Writing about changes to code should happen close in time to the changes themselves. Scriv encourages writing fragment files to be committed when you commit your code changes.
- How you describe a change depends on who you are describing it for. You may need multiple descriptions of the same change. Scriv encourages writing changelog entries directly, rather than copying text from commit messages or pull requests.
- The changelog file in the repo should be the source of truth. The information can also be published elsewhere, like GitHub releases.
- Different projects have different needs; flexibility is a plus. Scriv doesn't assume any particular issue tracker or packaging system, and allows either .rst or .md files.

2.1.1 Other tools

Scriv is not the first tool to help manage changelogs, there have been many. None fully embodied scriv's philopsophy.

Tools most similar to scriv:

- towncrier: built for Twisted, with some unusual specifics: fragment type is the file extension, issue numbers in the file name. Only .rst files.
- blurb: built for CPython development, specific to their workflow: issue numbers from bugs.python.org, only .rst files.
- setuptools-changelog: particular to Python projects (uses a setup.py command), and only supports .rst files.
- gitchangelog: collects git commit messages into a changelog file.

Tools that only read GitHub issues, or only write GitHub releases:

- Chronicler: a web hook that watched for merged pull requests, then appends the pull request message to the most recent draft GitHub release.
- fastrelease: reads information from GitHub issues, and writes GitHub releases.
- Release Drafter: adds text from merged pull requests to the latest draft GitHub release.

Other release note tools:

- reno: built for Open Stack. It stores changelogs forever as fragment files, only combining for publication.

2.2 Concepts

2.2.1 Fragments

Fragments are files describing your latest work, created by the "*scriv create*" command. The files are created in the changelog.d directory (settable with *fragment_directory*). Typically, they are committed with the code change itself, then later aggregated into the changelog file with "*scriv collect*".

2.2.2 Categories

Changelog entries can be categorized, for example as additions, fixes, removals, and breaking changes. The list of categories is settable with the *categories* setting.

If you are using categories in your project, new fragments will be pre-populated with all the categories, commented out. While editing the fragment, you provide your change information in the appropriate category. When the fragments are collected, they are grouped by category into a single changelog entry.

You can choose not to use categories by setting the *categories* setting to empty.

2.2.3 Entries

Fragments are collected into changelog entries with the "*scriv collect*" command. The fragments are combined in each category, in chronological order. The entry is given a header with version and date.

2.3 Commands

2.3.1 scriv create

```
$ scriv create --help
Usage: scriv create [OPTIONS]

  Create a new changelog fragment.

Options:
  --add / --no-add     'git add' the created file.
  --edit / --no-edit   Open the created file in your text editor.
  -v, --verbosity LVL  Either CRITICAL, ERROR, WARNING, INFO or DEBUG
  --help               Show this message and exit.
```

The create command creates new *fragments*.

File creation

Fragments are created in the changelog.d directory. The name of the directory can be configured with the *fragment_directory* setting.

The file name starts with the current date and time, so that entries can later be collected in chronological order. To help make the files understandable, the file name also includes the creator's git name, and the branch name you are working on. "Main" branch names aren't included, to cut down on uninteresting noise. The branch names considered uninteresting are settable with the *main_branches* setting.

The initial contents of the fragment file are populated from the *new_fragment_template* template. The format is either reStructuredText or Markdown, selectable with the *format* setting.

The default new fragment templates create empty sections for each *category*. Uncomment the one you want to use, and create a bullet for the changes you are describing. If you need a different template for new fragments, you can create a Jinja template and name it in the *new_fragment_template* setting.

Editing

If `--edit` is provided, or if `scriv.create.edit` is set to true in your *git settings*, scriv will launch an editor for you to edit the new fragment. Scriv uses the same editor that git launches for commit messages.

The format of the fragment should be sections for the categories, with bullets for each change. The file is re-parsed when it is collected, so the specifics of things like header underlines don't have to match the changelog file, that will be adjusted later.

Once you save and exit the editor, scriv will continue working on the file. If the file is empty because you removed all of the non-comment content, scriv will stop.

Adding

If `--add` is provided, or if `scriv.create.add` is set to true in your *git settings*, scriv will "git add" the new file so that it is ready to commit.

2.3.2 scriv collect

```
$ scriv collect --help
Usage: scriv collect [OPTIONS]

  Collect and combine fragments into the changelog.

Options:
  --add / --no-add     'git add' the updated changelog file and removed
                       fragments.
  --edit / --no-edit   Open the changelog file in your text editor.
  --title TEXT         The title text to use for this entry.
  --keep               Keep the fragment files that are collected.
  --version TEXT       The version name to use for this entry.
  -v, --verbosity LVL  Either CRITICAL, ERROR, WARNING, INFO or DEBUG
  --help               Show this message and exit.
```

The collect command aggregates all the current fragments into the changelog file.

Entry Creation

All of the .rst or .md files in the fragment directory are read, parsed, and re-assembled into a changelog entry. The entry's title is determined by the *entry_title_template* setting. The default uses the version string (if one is specified in the *version* setting) and the current date.

Instead of using the title template, you can provide an exact title to use for the new entry with the `--title` option.

The output file is specified by the *output_file* setting. Scriv looks in the file for a special marker (usually in a comment) to determine where to insert the new entry. The marker is "scriv-insert-here", but can be changed with the *insert_marker* setting. Using a marker like this, you can have your changelog be just part of a larger README file. If there is no marker in the file, the new entry is inserted at the top of the file.

Fragment Deletion

The fragment files that are read will be deleted, because they are no longer needed. If you would prefer to keep the fragment files, use the `--keep` option.

Editing

If `--edit` is provided, or if `scriv.collect.edit` is set to true in your *git settings*, scriv will launch an editor for you to edit the changelog file. Mostly you shouldn't need to do this, but you might want to make some tweaks. Scriv uses the same editor that git launches for commit messages.

Adding

If `--add` is provided, or if `scriv.collect.add` is set to true in your *git settings*, scriv will "git add" the updates to the changelog file, and the fragment file deletions, so that they are ready to commit.

2.3.3 scriv github-release

```
$ scriv github-release --help
Usage: scriv github-release [OPTIONS]

  Create GitHub releases from the changelog.

  Only the most recent changelog entry is used, unless --all is provided.

Options:
  --all                Use all of the changelog entries.
  --dry-run            Don't post to GitHub, just show what would be done.
  --repo TEXT          The GitHub repo (owner/reponame) to create the
                       release in.
  -v, --verbosity LVL  Either CRITICAL, ERROR, WARNING, INFO or DEBUG
  --help               Show this message and exit.
```

The `github-release` command reads the changelog file, parses it into entries, and then creates or updates GitHub releases to match. Only the most recent changelog entry is used, unless `--all` is provided.

An entry must have a version number in the title, and that version number must correspond to a git tag. For example, this changelog entry with the title `v1.2.3 -- 2022-04-06` will be processed and the version number will be "v1.2.3". If there's a "v1.2.3" git tag, then the entry is a valid release. If there's no detectable version number in the header, or there isn't a git tag with the same number, then the entry can't be created as a GitHub release.

This command is independent of the other commands. It can be used with a hand-edited changelog file that wasn't created with scriv.

For writing to GitHub, you need a GitHub personal access token, either stored in your .netrc file, or in the GITHUB_TOKEN environment variable.

The GitHub repo will be determined by examining the git remotes. If there is just one GitHub repo in the remotes, it will be used to create the release. You can explicitly specify a repo in `owner/reponame` form with the `--repo=` option if needed.

If your changelog file is in reStructuredText format, you will need pandoc 2.11.2 or later installed for the command to work.

2.4 Configuration

Scriv tries hard to be adaptable to your project's needs. Many aspects of its behavior can be customized with a settings file.

2.4.1 Files read

Scriv will read settings from any of these files:

- setup.cfg
- tox.ini
- pyproject.toml
- scriv.ini in the fragment directory ("changelog.d/" by default)

In .ini or .cfg files, scriv will read settings from a section named either `[scriv]` or `[tool.scriv]`.

A .toml file will only be read if the tomli or tomllib modules is available. You can install scriv with the `[toml]` extra to install tomli, or tomllib is available with Python 3.11 or greater. In a .toml file, settings will only be read from the `[tool.scriv]` section.

All of the possible files will be read, and settings will cascade. So for example, setup.cfg can set the fragment directory to "scriv.d", then "scriv.d/scriv.ini" will be read.

2.4.2 Settings Syntax

Settings use the usual syntax, but with some extra features:

- A prefix of `file:` reads the setting from a file.
- A prefix of `literal:` reads a literal data from a source file.
- Value substitutions can make a setting depend on another setting.

File Prefix

A `file:` prefix means the setting value is a file name, and the actual setting value will be read from that file. If the file name has path separators, it is relative to the current directory. If it doesn't have path separators, then it is either in the fragment directory (changelog.d by default), or one of the built-in provided templates.

Scriv provides two built-in templates:

- `new_fragment.md.j2`: The default Jinja template for new Markdown fragments:

```
<!--
A new scriv changelog fragment.

Uncomment the section that is right (remove the HTML comment wrapper).
-->

{% for cat in config.categories -%}
<!--
### {{ cat }}

- A bullet item for the {{ cat }} category.

-->
{% endfor -%}
```

- `new_fragment.rst.j2`: The default Jinja template for new reStructuredText fragments:

```
.. A new scriv changelog fragment.
{% if config.categories -%}
..
.. Uncomment the header that is right (remove the leading dots).
..
{% for cat in config.categories -%}
.. {{ cat }}
.. {{ config.rst_header_chars[1] * (cat|length) }}
..
.. - A bullet item for the {{ cat }} category.
..
{% endfor -%}
{% else %}
- A bullet item for this fragment. EDIT ME!
{% endif -%}
```

Literal Prefix

A `literal:` prefix means the setting value will be a literal string read from a source file. The setting provides a file name and value name separated by colons:

```
[scriv]
version = literal: myproj/__init__.py: __version__
```

In this case, the file `myproj/__init__.py` will be read, and the `__version__` value will be found and used as the version setting.

Currently Python, TOML and YAML files are supported for literals, but other syntaxes can be supported in the future.

When using a TOML file, the value is specified using periods to separate the sections and key names:

```
[scriv]
version = literal: pyproject.toml: project.version
```

In a YAML file, use periods in the value name to access dictionary keys:

```
[scriv]
version = literal: galaxy.yaml: myproduct.versionString
```

Value Substitution

The chosen fragment format can be used in settings by referencing `${config:format}` in the setting. For example, the default template for new fragments depends on the format because the default setting is:

```
new_fragment_template = file: new_fragment.${config:format}.j2
```

2.4.3 Settings

These are the specifics about all of the settings read from the configuration file.

categories

Categories to use as headings for changelog items. See *Categories*.

Default: `Removed, Added, Changed, Deprecated, Fixed, Security`

end_marker

A marker string indicating where in the changelog file the changelog ends.

Default: `scriv-end-here`

entry_title_template

The Jinja template to use for the entry heading text for changelog entries created by "*scriv collect*".

Default: A combination of version (if specified) and date.

format

The format to use for fragments and for the output changelog file. Can be either "rst" or "md".

Default: `rst`

fragment_directory

The directory for fragments. This directory must exist, it will not be created.

Default: `changelog.d`

ghrel_template

The template to use for GitHub releases created by the `scriv github-release` command.

The extracted Markdown text is available as `{{body}}`. You must include this to use the text from the changelog file. The version is available as `{{version}}`.

The data for the release is available in a `{{release}}` object, including `{{release.prerelease}}`. It's a boolean, true if this is a pre-release version.

The scriv configuration is available in a `{{config}}` object.

Default: `{{body}}`

insert_marker

A marker string indicating where in the changelog file new entries should be inserted.

Default: `scriv-insert-here`

main_branches

The branch names considered uninteresting to use in new fragment file names.

Default: `master, main, develop`

md_header_level

A number: for Markdown changelog files, this is the heading level to use for the entry heading.

Default: `1`

new_fragment_template

The Jinja template to use for new fragments.

Default: `file: new_fragment.${config:format}.j2`

output_file

The changelog file updated by "*scriv collect*".

Default: `CHANGELOG.${config:format}`

rst_header_chars

Two characters: for reStructuredText changelog files, these are the two underline characters to use. The first is for the heading for each changelog entry, the second is for the category sections within the entry.

Default: `=-`

skip_fragments

A glob pattern for files in the fragment directory that should not be collected.

Default: `README.*`

version

The string to use as the version number in the next header created by `scriv collect`. Often, this will be a `literal:` directive, to get the version from a string in a source file.

Default: (empty)

2.4.4 Per-User Git Settings

Some aspects of scriv's behavior are configurable for each user rather than for the project as a whole. These settings are read from git.

These settings determine whether the "*scriv create*" and "*scriv collect*" commands will launch an editor, and "git add" the result:

- `scriv.create.edit`
- `scriv.create.add`
- `scriv.collect.edit`
- `scriv.collect.add`

All of these are either "true" or "false", and default to false. You can create these settings with git config commands, either in the current repo:

```
$ git config scriv.create.edit true
```

or globally for all of your repos:

```
$ git config --global scriv.create.edit true
```

2.5 Changelog

2.5.1 Unreleased

See the fragment files in the changelog.d directory.

2.5.2 1.2.0 — 2023-01-18

Added

- `scriv github-release` now has a `--repo=` option to specify which GitHub repo to use when you have multiple remotes.

Changed

- Improved the error messages from `scriv github-release` when a GitHub repo can't be identified among the git remotes.

2.5.3 1.1.0 — 2023-01-16

Added

- The `scriv github-release` command has a new setting, `ghrel_template`. This is a template to use when building the release text, to add text before or after the Markdown extracted from the changelog.
- The `scriv github-release` command now has a `--dry-run` option to show what would happen, without posting to GitHub.

Changed

- File names specified for `file:` settings will be interpreted relative to the current directory if they have path components. If the file name has no slashes or backslashes, then the old behavior remains: the file will be found in the fragment directory, or as a built-in template.
- All exceptions raised by Scriv are now ScrivException.

Fixed

- Parsing changelogs now take the *insert-marker* setting into account. Only content after the insert-marker line is parsed.
- More internal activities are logged, to help debug operations.

2.5.4 1.0.0 — 2022-12-03

Added

- Now literal configuration settings can be read from YAML files. Closes issue 69. Thanks, Florian Küpper.

Fixed

- Fixed truncated help summaries by shortening them, closing issue 63.

2.5.5 0.17.0 — 2022-09-18

Added

- The `collect` command now has a `--title=TEXT` option to provide the exact text to use as the title of the new changelog entry. Finishes issue 48.

Changed

- The `github_release` command now only considers the top-most entry in the changelog. You can use the `--all` option to continue the old behavior of making or updating GitHub releases for all of the entries.

  This change makes it easier for projects to start using scriv with an existing populated changelog file.

  Closes issue 57.

Fixed

- If there were no fragments to collect, *scriv collect* would make a new empty section in the changelog. This was wrong, and is now fixed. Now the changelog remains unchanged in this case. Closes issue 55.
- The `github-release` command will now issue a warning for changelog entries that have no version number. These can't be made into releases, so they are skipped. (issue 56).
- `scriv collect` will end with an error now if the version number would duplicate a version number on an existing changelog entry. Fixes issue 26.

2.5.6 0.16.0 — 2022-07-24

Added

- The `github_release` command will use a GitHub personal access token stored in the GITHUB_TOKEN environment variable, or from a .netrc file.

Fixed

- The github_release command was using *git tags* as a command when it should have used *git tag*.
- Anchors in the changelog were being included in the previous sections when creating GitHub releases. This has been fixed, closing issue 53.

2.5.7 0.15.2 — 2022-06-18

Fixed

- Quoted commands failed, so we couldn't determine the GitHub remote.

2.5.8 0.15.1 — 2022-06-18

Added

- Added docs for `scriv github-release`.

Fixed

- Call pandoc properly on Windows for the github_release command.

2.5.9 0.15.0 — 2022-04-24

Removed

- Dropped support for Python 3.6.

Added

- The *github-release* command parses the changelog and creates GitHub releases from the entries. Changed entries will update the corresponding release.
- Added a `--version` option.

Changed

- Parsing of fragments now only attends to the top-level section headers, and includes nested headers instead of splitting on all headers.

2.5.10 0.14.0 — 2022-03-23

Added

- Add an anchor before each version section in the output of `scriv collect` so URLs for the sections are predictable and stable for each new version (Fixes issue 46). Thanks Abhilash Raj and Rodrigo Girão Serrão.

Fixed

- Markdown fragments weren't combined properly. Now they are. Thanks Rodrigo Girão Serrão.

2.5.11 0.13.0 — 2022-01-23

Added

- Support finding version information in TOML files (like `pyproject.toml`) using the `literal` configuration directive. Thanks, Kurt McKee

2.5.12 0.12.0 — 2021-07-28

Added

- Fragment files in the fragment directory will be skipped if they match the new configuration value `skip_fragments`, a glob pattern. The default value is "README.*". This lets you put a README.md file in that directory to explain its purpose, as requested in issue 40.

Changed

- Switched from "toml" to "tomli" for reading TOML files.

Fixed

- Setting `format=md` didn't properly cascade into other default settings, leaving you with RST settings that needed to be explicitly overridden (issue 39). This is now fixed.

2.5.13 0.11.0 — 2021-06-22

Added

- A new poorly documented API is available. See the Scriv, Changelog, and Fragment classes in the scriv.scriv module.

Changed

- Python 3.6 is now the minimum supported Python version.

Fixed

- The changelog is now always written as UTF-8, regardless of the default encoding of the system. Thanks, Hei (yhlam).

2.5.14 0.10.0 — 2020-12-27

Added

- Settings can now be read from a pyproject.toml file. Install with the "[toml]" extra to be sure TOML support is available. Closes issue 9.
- Added the Philosophy section of the docs.

Changed

- The default entry header no longer puts the version number in square brackets: this was a misunderstanding of the keepachangelog formatting.
- Respect the existing newline style of changelog files. (#14) This means that a changelog file with Linux newlines on a Windows platform will be updated with Linux newlines, not rewritten with Windows newlines. Thanks, Kurt McKee.

Fixed

- Support Windows' directory separator (\) in unit test output. (#15) This allows the unit tests to run in Windows environments. Thanks, Kurt McKee.
- Explicitly specify the directories and files that Black should scan. (#15) This prevents Black from scanning every file in a virtual environment. Thanks, Kurt McKee.
- Using "literal:" values in the configuration file didn't work on Python 3.6 or 3.7, as reported in issue 18. This is now fixed.

2.5.15 0.9.2 — 2020-08-29

- Packaging fix.

2.5.16 0.9.0 — 2020-08-29

Added

- Markdown format is supported, both for fragments and changelog entries.
- Fragments can be mixed (some .rst and some .md). They will be collected and output in the format configured in the settings.
- Documentation.
- "python -m scriv" now works.

Changed

- The version number is displayed in the help message.

2.5.17 0.8.1 — 2020-08-09

Added

- When editing a new fragment during "scriv create", if the edited fragment has no content (only comments or blank lines), then the create operation will be aborted, and the file will be removed. (Closes issue 2.)

Changed

- If the fragment directory doesn't exist, a simple direct message is shown, rather than a misleading FileNotFound error (closes issue 1).

Fixed

- When not using categories, comments in fragment files would be copied to the changelog file (issue 3). This is now fixed.
- RST syntax is better understood, so that hyperlink references and directives will be preserved. Previously, they were mistakenly interpreted as comments and discarded.

2.5.18 0.8.0 — 2020-08-04

Added

- Added the *collect* command.
- Configuration is now read from setup.cfg or tox.ini.
- A new configuration setting, rst_section_char, determines the character used in the underlines for the section headings in .rst files.
- The *new_entry_template* configuration setting is the name of the template file to use when creating new entries. The file will be found in the *fragment_directory* directory. The file name defaults to `new_entry.FMT.j2`. If the file doesn't exist, an internal default will be used.
- Now the collect command also includes a header for the entire entry. The underline is determined by the "rst_header_char" settings. The heading text is determined by the "header" setting, which defaults to the current date.
- The categories list in the config can be empty, meaning entries are not categorized.
- The create command now accepts –edit (to open the new entry in your text editor), and –add (to "git add" the new entry).
- The collect command now accepts –edit (to open the changelog file in an editor after the new entries have been collected) and –add (to git-add the changelog file and git rm the entries).
- The names of the main git branches are configurable as "main_branches" in the configuration file. The default is "master", "main", and "develop".
- Configuration values can now be read from files by prefixing them with "file:". File names will be interpreted relative to the changelog.d directory, or will be found in a few files installed with scriv.
- Configuration values can interpolate the currently configured format (rst or md) with "${config:format}".
- The default value for new templates is now "file: new_entry.${config:format}.j2".
- Configuration values can be read from string literals in Python code with a "literal:" prefix.
- "version" is now a configuration setting. This will be most useful when used with the "literal:" prefix.
- By default, the title of collected changelog entries includes the version if it's defined.
- The collect command now accepts a `--version` option to set the version name used in the changelog entry title.

Changed

- RST now uses minuses instead of equals.
- The *create* command now includes the time as well as the date in the entry file name.
- The –delete option to collect is now called –keep, and defaults to False. By default, the collected entry files are removed.
- Created file names now include the seconds from the current time.
- "scriv create" will refuse to overwrite an existing entry file.
- Made terminology more uniform: files in changelog.d are "fragments." When collected together, they make one changelog "entry."

- The title text for the collected changelog entry is now created from the "entry_title_template" configuration setting. It's a Jinja2 template.
- Combined the rst_header_char and rst_section_char settings into one: rst_header_chars, which much be exactly two characters.
- Parsing RST fragments is more flexible: the sections can use any valid RST header characters for the underline. Previously, it had to match the configured RST header character.

Fixed

- Fragments with no category header were being dropped if categories were in use. This is now fixed. Uncategorized fragments get sorted before any categorized fragments.

2.5.19 0.1.0 — 2019-12-30

- Doesn't really do anything yet.