civic-scraper

unknown

Jun 20, 2023

CONTENTS

Tools for downloading agendas, minutes and other documents produced by local government

CHAPTER

ONE

DOCUMENTATION

1.1 Getting started

Install the library from the Python Package Index.

```
pipenv install civic-scraper
```

Upon installation, you should have access to the `civic-scraper` tool on the command line:

```
pipenv run civic-scraper --help
```

… and start scraping from the command line:

```
pipenv run civic-scraper scrape --download --url http://nc-nashcounty.civicplus.com/
↪AgendaCenter
```

Or in a script:

```
from civic_scraper.platforms import CivicPlusSite

url = "http://nc-nashcounty.civicplus.com/AgendaCenter"
site = CivicPlusSite(url)
site.scrape(download=True)
```

Note: There are many more options for customizing scrapes, especially by date range. Check out the *Usage* docs for details. See the *Getting started* docs to configure the download location.

1.2 Usage

1.2.1 Overview

civic-scraper provides a command-line tool and underlying Python library that can be used to fetch metadata about government documents and help download those documents.

The command-line tool makes it easy to get started scraping for basic use cases, while the Python library offers a wider range of options for use in custom scripts.

Government agendas and other files downloaded by *civic-scraper* are saved to a a standard – but *configurable* – location in the user's home directory (`~/.civic-scraper` on Linux/Mac).

Below are more details on using the *Command line* as well as writing *Custom scripts*.

Note: civic-scraper currently supports scraping of five software platforms: Civic Clerk, Civic Plus, Granicus, Legistar and PrimeGov.

1.2.2 Find a site to scrape

Before you can start scraping government documents, you must first pinpoint URLs for one or more agencies of interest. Alternatively, you may want to review our lists of known Civic Plus sites or Legistar sites to see if any agencies in your area use one of these platforms.

In addition to Civic Plus and Legistar, *civic-scraper* currently supports Civic Clerk, Granicus and PrimeGov.

If your target agency uses one of these platforms, you should be able to scrape the site by writing a Python script that uses the appropriate platform scraper class.

If your agency site is not currently supported, you can try reaching out to us to see if the platform is on our development roadmap. We also welcome *open-source contributions* if you want to add support for a new platform.

1.2.3 Command line

Once you *install* *civic-scraper* and *find a site to scrape*, you're ready to begin using the command-line tool.

Note: To test drive examples below, you should replace <site URL> with a URL to a Civic Plus site, e.g. http://nc-nashcounty.civicplus.com/AgendaCenter.

Getting help

civic-scraper provides a scrape subcommand as the primary way to fetch metadata and files from government sites. You can use the tool's --help flag to get details on the available options:

```
civic-scraper scrape --help
```

Basic usage

By default, *civic-scraper* checks a site for meetings that occur on the current day and generates a metadata CSV listing information about any available meeting agendas or minutes:

```
# Scrape current day and generate metadata CSV
civic-scraper scrape --url <site URL>
```

Download documents

civic-scraper does not automatically download agendas or minutes by default since, depending on the *time period of the scrape* and size of the documents, this could involve a large quantity of data.

You must explicitly tell *civic-scraper* to download documents by using the `--download` flag, which will fetch and save agendas/minutes to *civic-scraper*'s *cache directory*:

```
civic-scraper scrape --download --url <site URL>
```

Scrape by date

civic-scraper provides the ability to set a date range to support scraping documents from meetings in the past:

```
# Scrape docs from meetings in January 2020
civic-scraper scrape \
  --start-date=2020-01-01 \
  --end-date=2020-01-31 \
  --url <site URL>
```

Scrape multiple sites

If you need to scrape more than one site at a time, you can supply a CSV containing URLs to *civic-scraper*.

The input CSV must store site URLs in a column called `url`, similar to the list of known sites for the Civic Plus platform.

Let's say we have a *ca_examples.csv* with two agencies in California:

```
state,url
ca, https://ca-alpinecounty.civicplus.com/AgendaCenter
ca, https://ca-anaheim.civicplus.com/AgendaCenter
```

You can scrape both sites by supplying the CSV's path to the `--urls-file` flag:

```
# Scrape current day for URLs listed in CSV (should contain "url" field)
civic-scraper scrape --urls-file ca_examples.csv
```

Store scraping artifacts

As part of the scraping process, *civic-scraper* acquires "intermediate" file artifacts such as HTML pages with links to meeting agendas and minutes.

We believe it's important to keep such file artifacts for the sake of transparency and reproducibility.

Use the `--cache` flag to store these files in the *civic-scraper cache directory*:

```
civic-scraper scrape --cache  --url <site URL>
```

Putting it all together

The command-line options mentioned above can be used in tandem (with the exception of `--url` and `--urls-file`, which are mutually exclusive).

For example, the below command:

```
civic-scraper scrape \
  --cache \
  --download \
  --start-date=2020-01-01 \
  --end-date=2020-01-31 \
  --url <site URL>
```

would performing the following actions:

- Generate a [Metadata CSV] on available documents for meetings in January 2020
- *Download* agendas and minutes for meetings in the specified date range
- *Cache* the HTML of search results pages containing links to agendas/minutes

1.2.4 Custom scripts

civic-scraper provides an importable Python package for users who are comfortable creating their own scripts. The Python package provides access to a wider variety of features for added flexibility and support for more advanced scenarios (e.g controlling the location of downloaded files or avoiding download of excessively large files).

Note: In order to use *civic-scraper* in a script, you must install the package and import one of the platform scraper classes. In the examples below, we use the `CivicPlusSite` class. See the platforms folder on GitHub for other available platform classes.

Site classes may support slightly different interfaces/features due to differences in features on each platform.

It's a good idea to review the docstrings and methods for a class before attempting to use it.

Scrape metadata

Once you *install* *civic-scraper* and *find a site to scrape*, you're ready to begin using the `civic_scraper` Python package.

Note: Below we use East Palo Alto, CA as an example. More agencies can be found in the list of known sites for the Civic Plus platform.

Create an instance of `CivicPlusSite` by passing it the URL for an agency's CivicPlus Agenda Center site. Then call the `scrape` method:

```python
from civic_scraper.platforms import CivicPlusSite
url = 'https://ca-eastpaloalto.civicplus.com/AgendaCenter'
site = CivicPlusSite(url)
assets_metadata = site.scrape()
```

Note: CivicPlusSite is an alias for more convenient import of the actual Civic Plus class located at civic_scraper.platforms.civic_plus.site.Site.

CivicPlusSite.scrape will automatically store downloaded assets in the *default cache directory*.

This location can be customized by *setting an environment variable* or by passing an instance of *civic_scraper.base.cache.Cache* to CivicPlusSite:

```
from civic_scraper.base.cache import Cache
from civic_scraper.platforms import CivicPlusSite

url = 'https://ca-eastpaloalto.civicplus.com/AgendaCenter'

# Change output dir to /tmp
site = CivicPlusSite(url, cache=Cache('/tmp'))
assets_metadata = site.scrape()
```

Export metadata to CSV

By default, CivicPlusSite.scrape returns an *AssetCollection* containing *Asset* instances.

The asset instances store metadata about specific meeting agendas and minutes discovered on the site.

To save a timestamped CSV containing metadata for available assets, call *AssetCollection.to_csv()* with a target output directory:

```
# Save metadata CSV
assets_metadata.to_csv('/tmp/civic-scraper/metadata')
```

Download assets

There are two primary ways to download file assets discovered by a scrape.

You can trigger downloads by passing download=True to CivicPlusSite.scrape:

```
site.scrape(download=True)
```

Or you can loop over the *Asset instances* in an *AssetCollection* and call *download()* on each with a target output directory:

```
assets_metadata = site.scrape()
for asset in assets_metadata:
    asset.download('/tmp/civic-scraper/assets')
```

Scrape by date

By default, scraping checks the site for meetings on the current day (based on a user's local time).

Scraping can be modified to capture assets from different date ranges by supplying the optional `start_date` and/or `end_date` arguments to `CivicPlusSite.scrape`.

Their values must be strings of the form `YYYY-MM-DD`:

```
# Scrape info from January 1-30, 2020
assets_metadata = site.scrape(start_date='2020-01-01', end_date='2020-01-30')
```

Note: The above will *not* download the assets by default. See *download assets script* for details on saving the discovered files locally.

Advanced configuration

You can exercise more fine-grained control over the size and type of files to download using the `file_size` and `asset_list` arguments to `CivicPlusSite.scrape`:

```
# Download only minutes that are 20MB or smaller
site.scrape(
  download=True,
  file_size=20,
  asset_list=['minutes']
)
```

Here are more details on the parameters mentioned above:

- `file_size` - Limit downloads to files with max file size in megabytes.
- `asset_list` - Limit downloads to one or more [asset types] (described below in *Metadata CSV*). The default is to download all document types.

1.2.5 Metadata CSV

civic-scraper provides the ability to produce a CSV of metadata about agendas, minutes and other files discovered during a scrape. The file is automatically generated when using the *command line* and can be exported using `AssetCollection.to_csv` in the context of a *custom script*.

The generated file contains the following information:

- `url` (*str*) - The download link for an asset
- `asset_name` (*str*) - The title of an asset. Ex: City Council Special Budget Meeting - April 4, 2020
- `committee_name` (*str*) - The name of the committee that generated the asset. Ex: City Council
- `place` (*str*) - Name of the place associated with the asset (lowercased, punctuation removed). Ex: eastpaloalto
- `state_or_province` (*str*) - The lowercase two-letter abbreviation for the state or province associated with an asset
- `asset_type` (*str*) - One of the _`asset types` for meeting-related documents:
  - `agenda`

  - minutes
  - audio
  - video
  - agenda_packet - The exhibits and ancillary documents attached to a meeting agenda.
  - captions - The transcript of a meeting recording.
- meeting_date (*str*) - Date of meeting or blank if no meeting date given in the format YYYY-MM-DD.
- meeting_time (*str*) - Time of meeting or blank if no time given.
- meeting_id (*str*) - A unique meeting ID assigned to the record.
- scraped_by (*str*) - Version of *civic-scraper* that produced the asset. Ex: civicplus_v0.1.0
- content_type (*str*) - The MIME type of the asset. Ex: application/pdf
- content_length (*str*) - The size of the asset in bytes.

1.2.6 Changing the download location

By default, *civic-scraper* will store downloaded agendas, minutes and other files in a *default directory*.

You can *customize this location* by setting the CIVIC_SCRAPER_DIR environment variable.

1.3 Advanced configuration

1.3.1 Default cache directory

By default, files downloaded by the CLI tool and underlying Python library code will be saved to the .civic-scraper folder in the user's home directory.

On Linux/Mac systems, this will be ~/.civic-scraper/.

1.3.2 Customize cache directory

To use an alternate cache directory, set the below environment variable (e.g. in a ~/.bashrc or ~/.bash_profile configuration file):

```
export CIVIC_SCRAPER_DIR=/tmp/some_other_dir
```

1.4 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:

1.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/biglocalnews/civic-scraper/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.

Fix Bugs

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

Implement Features

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

Do Research

This project involves a fair bit of research, especially with respect to locating platforms and sites to scrape. Research jobs are great ways to get involved if you don't write code but still want to pitch in. Anything tagged with the "research" and "help wanted" labels on GitHub is fair game.

Write Documentation

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

Our official docs use restructuredText and Sphinx. To contribute documentation:

- Fork and clone this repo
- Create a virtual environment and perform the next steps inside an active venv
- `pip install -r requirements.txt` and `pip install -r requirements-dev.txt`
- Create a branch for your doc updates and start writing!
- Use `make docs` to build docs and/or use `make servedocs` commands to run a Sphinx server that displays doc pages and allows easier reloading of pages in browser
- Create a GitHub Pull Request once you're ready to send us your changes

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/biglocalnews/civic-scraper/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 :)

1.4.2 Get Started!

Ready to contribute? Here's how to set up `civic-scraper` for local development.

1. Fork the `civic-scraper` repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/civic-scraper.git
   ```

3. Install developement dependencies and your local copy of the code into a virtualenv and set up your fork for local development. There are numerous ways to create virtual environments in Python. Below uses the venv library built into recent Python versions:

   ```
   # Create a virtual env alongside the civic-scraper git repo
   python -m venv civic-scraper-env

   # Activate the virtual env
   source civic-scraper-env/bin/activate

   # Install dev requirements and the Python package into the venv
   cd civic-scraper/
   pip install -r requirements-dev.txt
   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 civic_scraper tests
   $ 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.

1.4.3 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, please be sure to review the docs and include necessary updates. For example, new classes, methods and functions should be documented.
3. The pull request should work for Python version 3.6 or higher. Check https://travis-ci.com/github/biglocalnews/civic-scraper/pull_requests and make sure that the tests pass for all supported Python versions.

1.5 Releasing

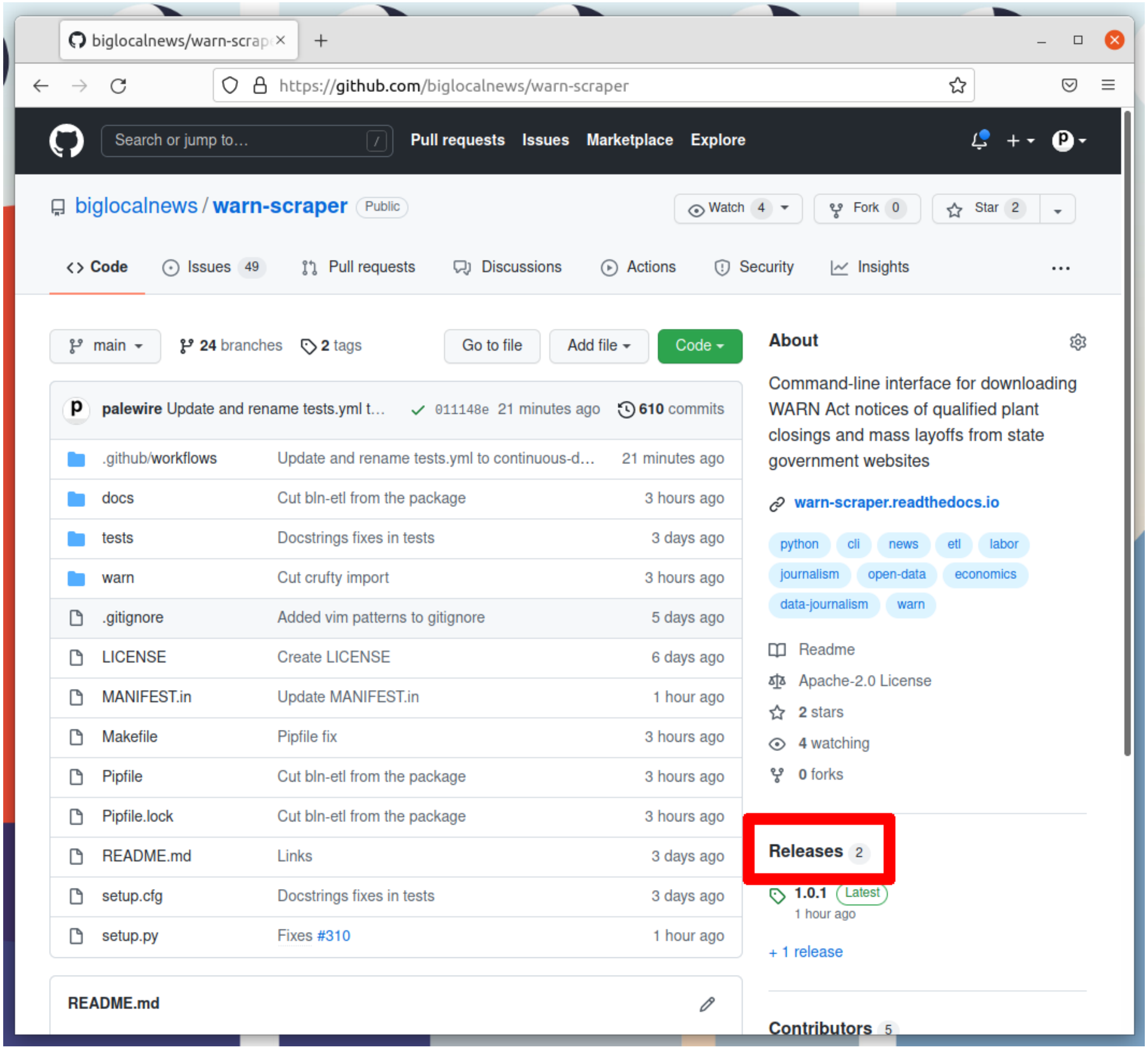
Our release process is automated as a continuous deployment via the GitHub Actions framework. The logic that governs the process is stored in the `workflows` directory.

That means that everything necessary to make a release can be done with a few clicks on the GitHub website. All you need to do is make a tagged release at biglocalnews/civic-scraper/releases, then wait for the computers to handle the job.

Here's how it's done, step by step. The screenshots are from a different repository, but the process is the same.

1.5.1 1. Go to the releases page

The first step is to visit our repository's homepage and click on the "releases" headline in the right rail.



palewire Update and rename tests.yml t… ✓ 011148e 21 minutes ago 610 commits

.github/workflows	Update and rename tests.yml to continuous-d…	21 minutes ago
docs	Cut bln-etl from the package	3 hours ago
tests	Docstrings fixes in tests	3 days ago
warn	Cut crufty import	3 hours ago
.gitignore	Added vim patterns to gitignore	5 days ago
LICENSE	Create LICENSE	6 days ago
MANIFEST.in	Update MANIFEST.in	1 hour ago
Makefile	Pipfile fix	3 hours ago
Pipfile	Cut bln-etl from the package	3 hours ago
Pipfile.lock	Cut bln-etl from the package	3 hours ago
README.md	Links	3 days ago
setup.cfg	Docstrings fixes in tests	3 days ago
setup.py	Fixes #310	1 hour ago

README.md

About

Command-line interface for downloading WARN Act notices of qualified plant closings and mass layoffs from state government websites

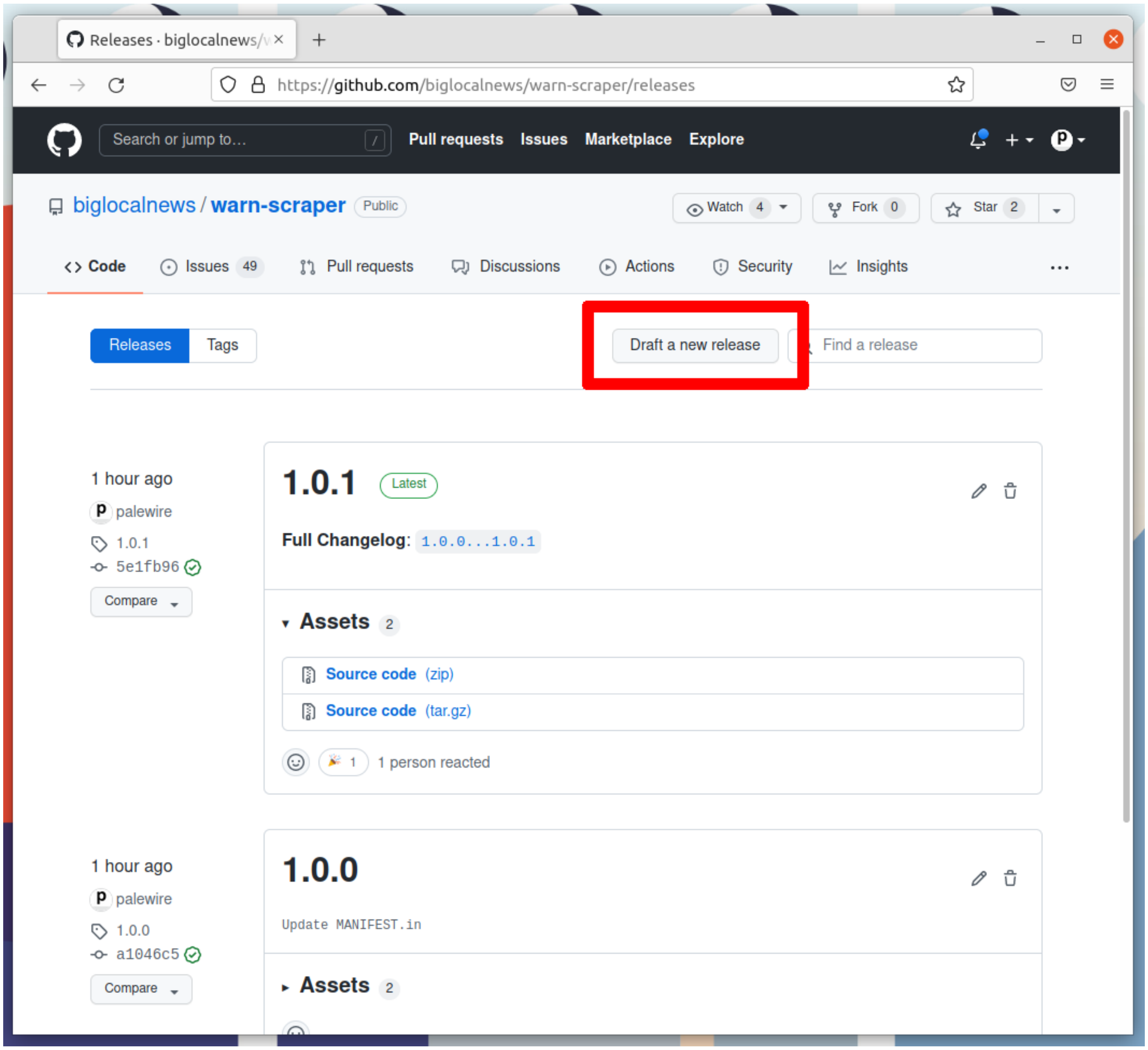
warn-scraper.readthedocs.io

python cli news etl labor journalism open-data economics data-journalism warn

Readme
Apache-2.0 License
2 stars
4 watching
0 forks

1.5.2 2. Click 'Draft a new release'

Note the number of the latest release. Click the "Draft a new release" button in the upper-right corner. If you don't see this button, you do not have permission to make a release. Only the maintainers of the repository are able to release new code.



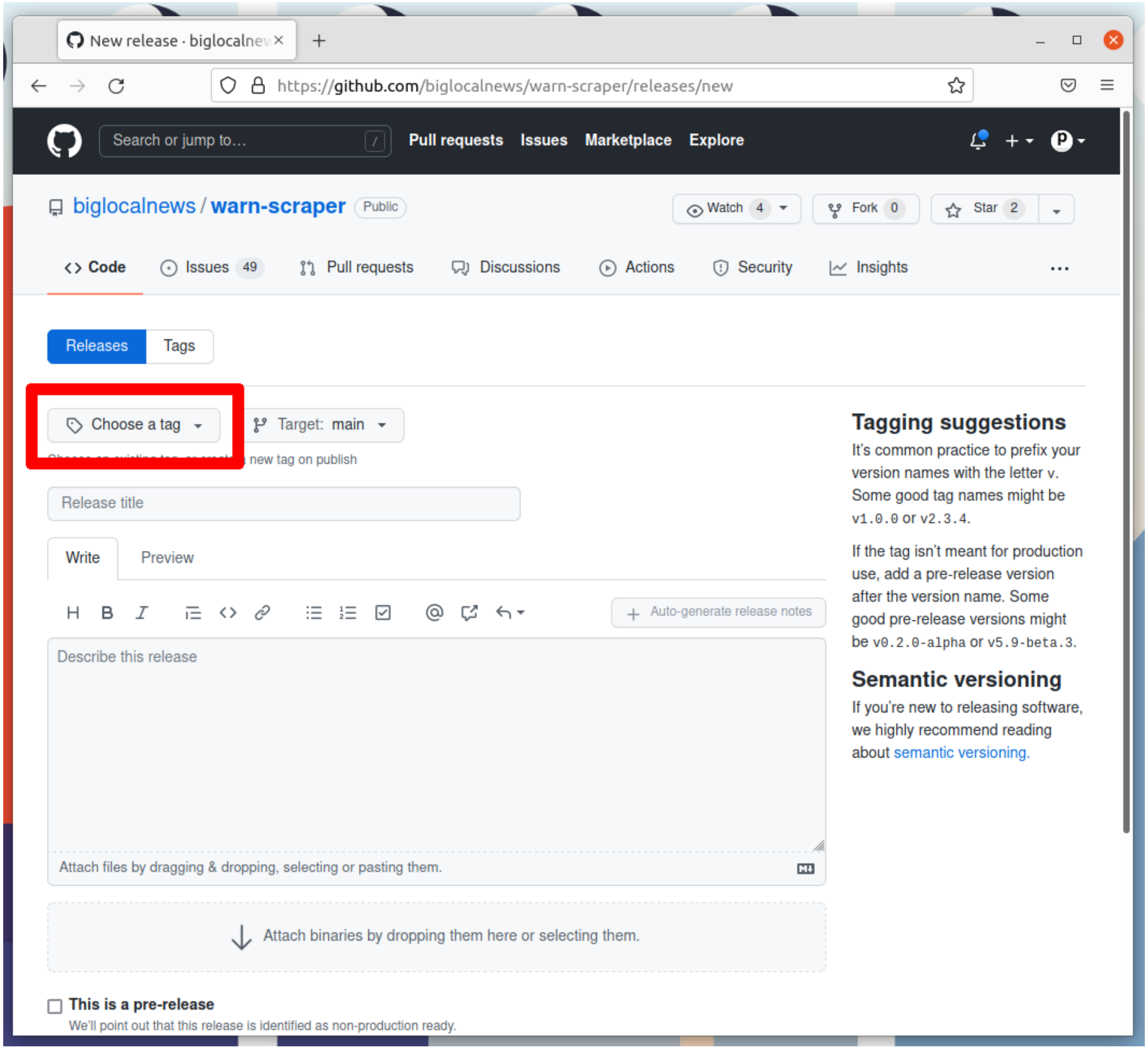
1.5.3 3. Create a new tag

Think about how big your changes are and decide if you're a major, minor or patch release.

All version numbers should feature three numbers separated by the periods, like `1.0.1`. If you're making a major release that isn't backwards compatible, the latest release's first number should go up by one. If you're making a minor release by adding a feature or major a large change, the second number should go up. If you're only fixing bugs or making small changes, the third number should go up.

If you're unsure, review the standards defined at semver.org to help make a decision. In the end don't worry about it too much. Our version numbers don't need to be perfect. They just need to be three numbers separated by periods.

Once you've settled on the number for your new release, click on the "Choose a tag" pull down.


New release · biglocalnew
https://github.com/biglocalnews/warn-scraper/releases/new
Search or jump to...
Pull requests
Issues
Marketplace
Explore
biglocalnews / warn-scraper
Public
Watch 4
Fork 0
Star 2
Code
Issues 49
Pull requests
Discussions
Actions
Security
Insights
Releases
Tags
Choose a tag
Target: main
new tag on publish
Release title
Write
Preview
Auto-generate release notes
Describe this release
Attach files by dragging & dropping, selecting or pasting them.
Attach binaries by dropping them here or selecting them.
This is a pre-release
We'll point out that this release is identified as non-production ready.

Tagging suggestions

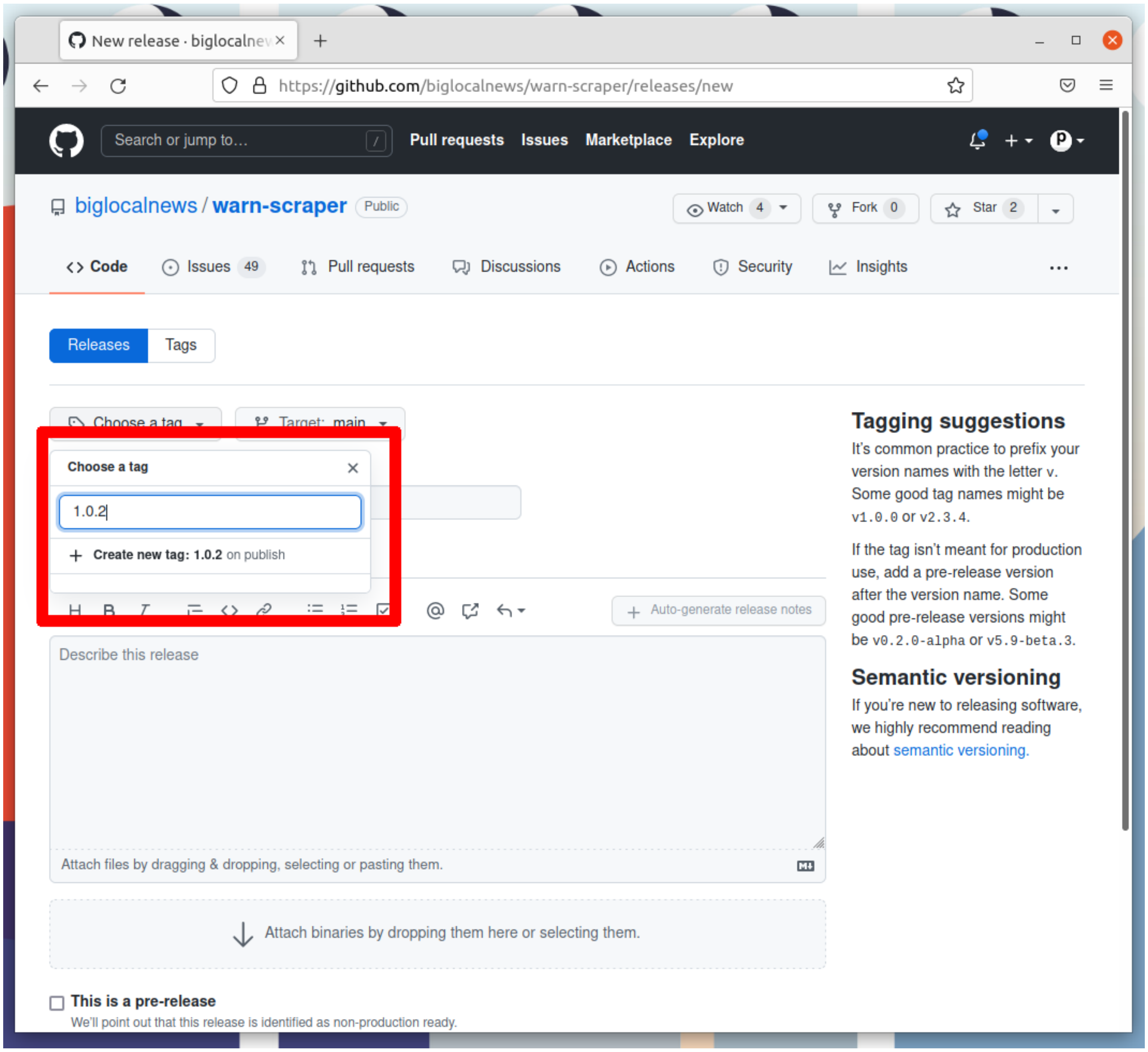
It's common practice to prefix your version names with the letter v. Some good tag names might be `v1.0.0` or `v2.3.4`.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be `v0.2.0-alpha` or `v5.9-beta.3`.

Semantic versioning

If you're new to releasing software, we highly recommend reading about semantic versioning.

Enter your version number into the box. Then click the "Create new tag" option that appears.



https://github.com/biglocalnews/warn-scraper/releases/new

1.0.2

Tagging suggestions

It's common practice to prefix your version names with the letter v. Some good tag names might be `v1.0.0` or `v2.3.4`.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be `v0.2.0-alpha` or `v5.9-beta.3`.

Semantic versioning

If you're new to releasing software, we highly recommend reading about semantic versioning.

Describe this release

Attach files by dragging & dropping, selecting or pasting them.

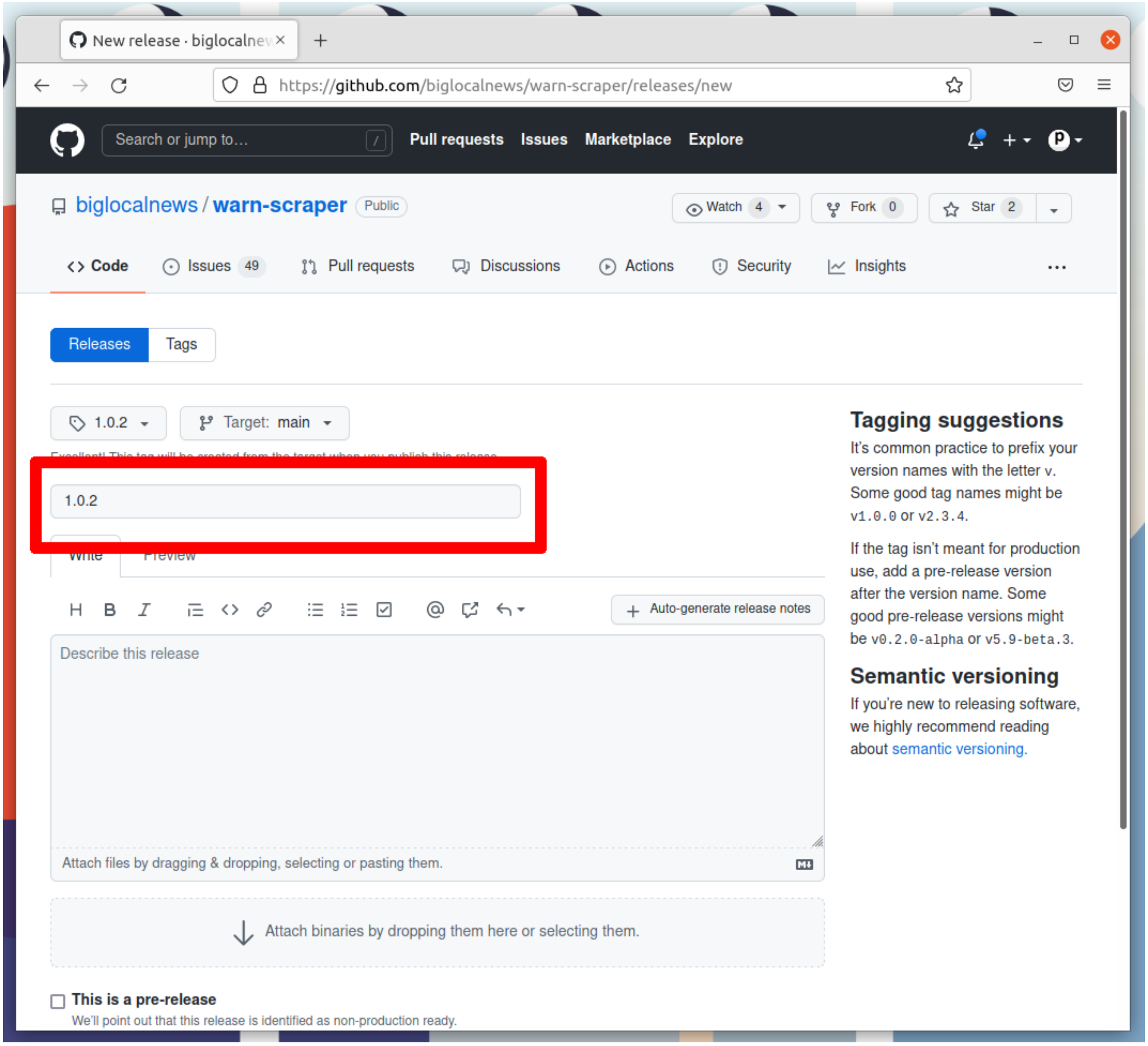
Attach binaries by dropping them here or selecting them.

□ **This is a pre-release**

We'll point out that this release is identified as non-production ready.

1.5.4 4. Name the release

Enter the same number into the "Release title" box.



New release · biglocalnew
https://github.com/biglocalnews/warn-scraper/releases/new
Search or jump to…
Pull requests
Issues
Marketplace
Explore
biglocalnews / warn-scraper
Public
Watch 4
Fork 0
Star 2
Code
Issues 49
Pull requests
Discussions
Actions
Security
Insights
Releases
Tags
1.0.2
Target: main
1.0.2
Write
Preview
Auto-generate release notes
Describe this release
Attach files by dragging & dropping, selecting or pasting them.
Attach binaries by dropping them here or selecting them.

Excellent! This tag will be created from the target when you publish this release.

Tagging suggestions

It's common practice to prefix your version names with the letter v. Some good tag names might be `v1.0.0` or `v2.3.4`.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be `v0.2.0-alpha` or `v5.9-beta.3`.

Semantic versioning

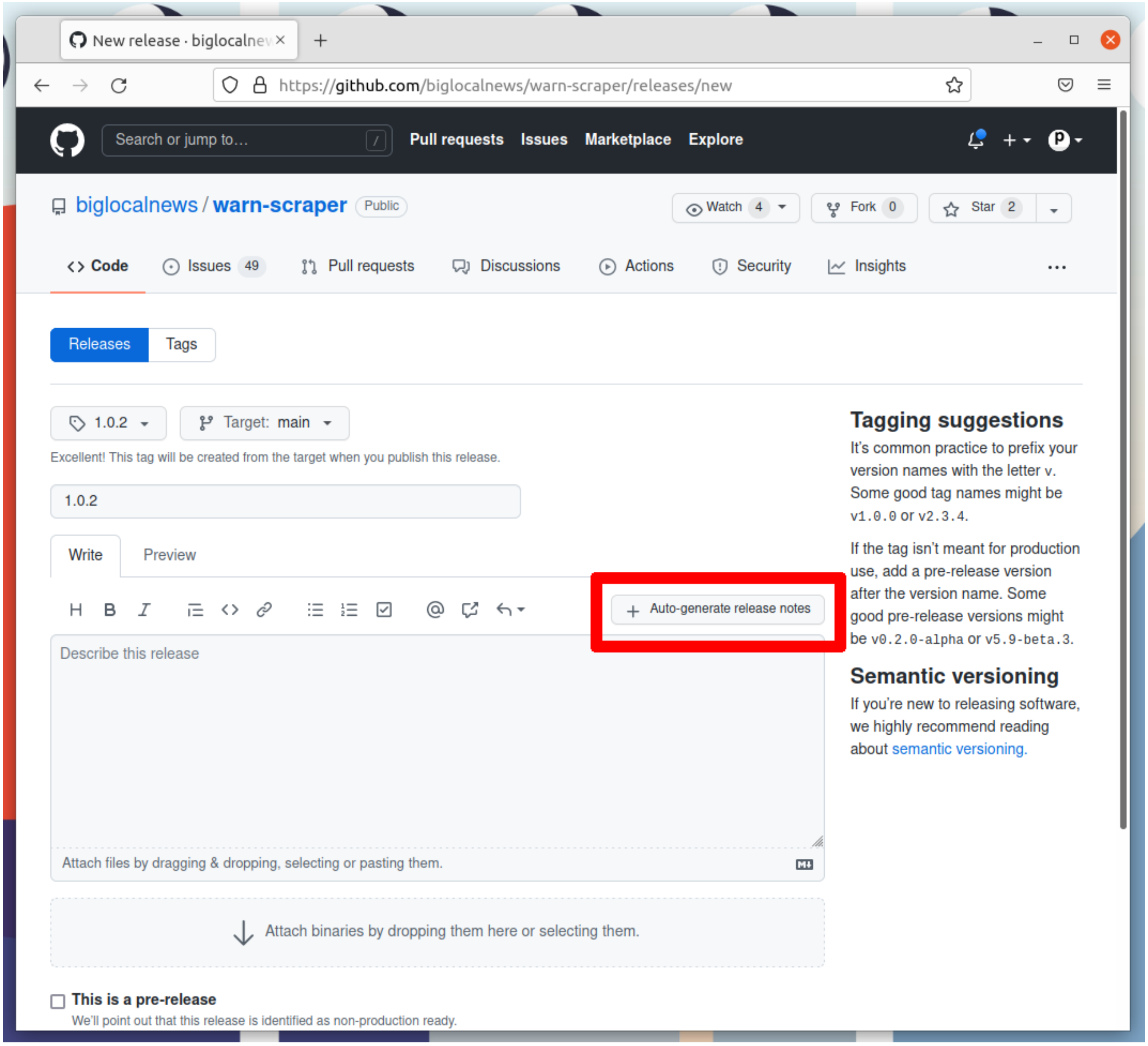
If you're new to releasing software, we highly recommend reading about semantic versioning.

□ **This is a pre-release**

We'll point out that this release is identified as non-production ready.

1.5.5 5. Auto-generate release notes

Click the "Auto-generate release notes" button in the upper right corner of the large description box.


New release · biglocalnev
https://github.com/biglocalnews/warn-scraper/releases/new
Search or jump to…
Pull requests
Issues
Marketplace
Explore
biglocalnews / warn-scraper
Public
Watch 4
Fork 0
Star 2
Code
Issues 49
Pull requests
Discussions
Actions
Security
Insights
Releases
Tags
1.0.2
Target: main
Write
Preview
Auto-generate release notes
Describe this release
Attach files by dragging & dropping, selecting or pasting them.
Attach binaries by dropping them here or selecting them.

Excellent! This tag will be created from the target when you publish this release.

1.0.2

Tagging suggestions

It's common practice to prefix your version names with the letter v. Some good tag names might be `v1.0.0` or `v2.3.4`.

If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be `v0.2.0-alpha` or `v5.9-beta.3`.

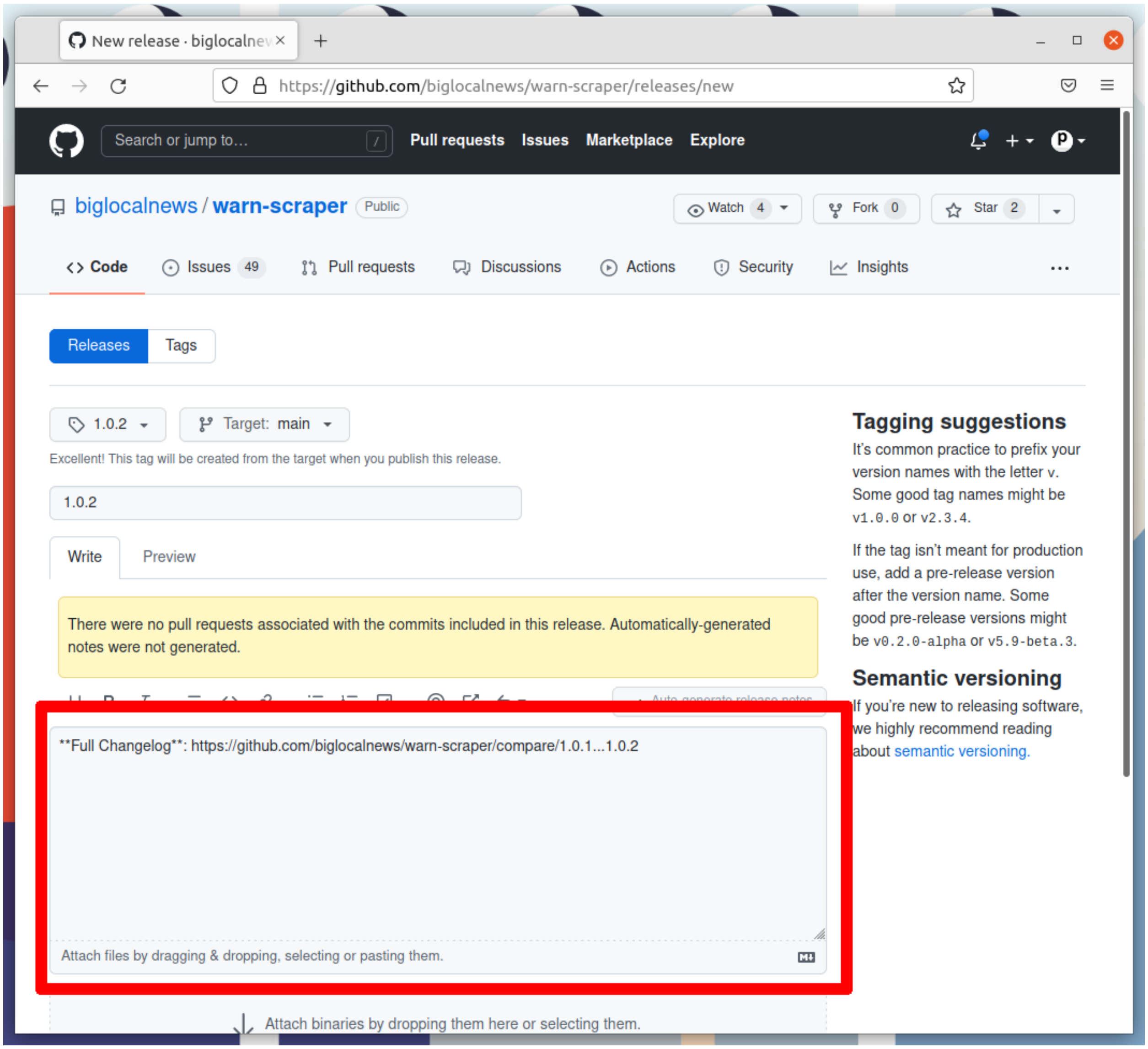
Semantic versioning

If you're new to releasing software, we highly recommend reading about semantic versioning.

□ **This is a pre-release**

We'll point out that this release is identified as non-production ready.

That should fill in the box below. What appears will depend on how many pull requests you've merged since the last release.


New release · biglocalnew
https://github.com/biglocalnews/warn-scraper/releases/new
Search or jump to…
Pull requests
Issues
Marketplace
Explore
biglocalnews / warn-scraper
Public
Watch 4
Fork 0
Star 2
Code
Issues 49
Pull requests
Discussions
Actions
Security
Insights
Releases
Tags
1.0.2
Target: main
Write
Preview
Attach files by dragging & dropping, selecting or pasting them.
Attach binaries by dropping them here or selecting them.

Excellent! This tag will be created from the target when you publish this release.

1.0.2

There were no pull requests associated with the commits included in this release. Automatically-generated notes were not generated.

Full Changelog: https://github.com/biglocalnews/warn-scraper/compare/1.0.1...1.0.2

Tagging suggestions

It's common practice to prefix your version names with the letter v. Some good tag names might be `v1.0.0` or `v2.3.4`.

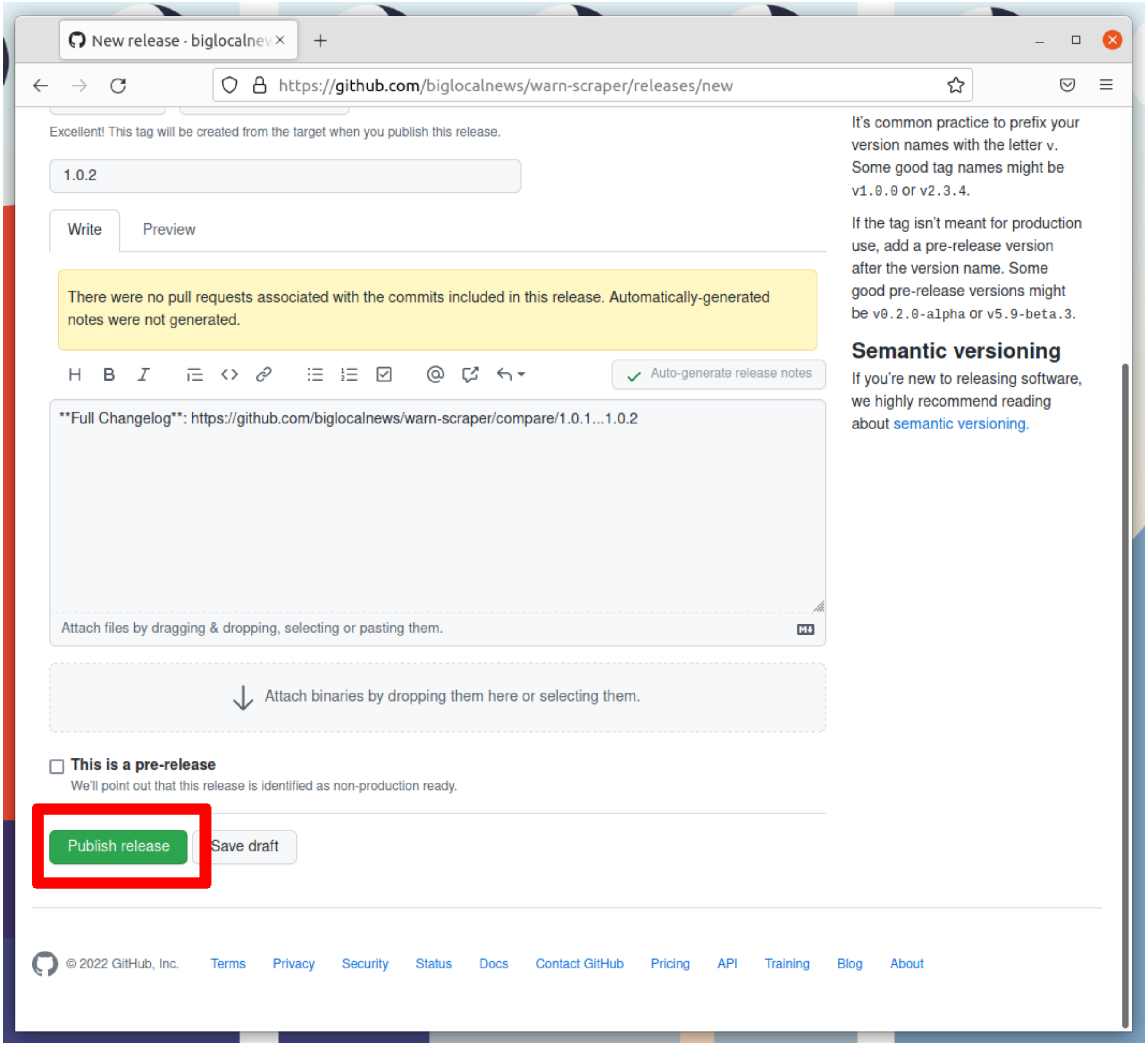
If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be `v0.2.0-alpha` or `v5.9-beta.3`.

Semantic versioning

If you're new to releasing software, we highly recommend reading about semantic versioning.

1.5.6 6. Publish the release

Click the green button that says "Publish release" at the bottom of the page.



Excellent! This tag will be created from the target when you publish this release.

1.0.2

There were no pull requests associated with the commits included in this release. Automatically-generated notes were not generated.

Full Changelog: https://github.com/biglocalnews/warn-scraper/compare/1.0.1...1.0.2

Attach files by dragging & dropping, selecting or pasting them.

Attach binaries by dropping them here or selecting them.

□ **This is a pre-release**
We'll point out that this release is identified as non-production ready.

It's common practice to prefix your version names with the letter v. Some good tag names might be `v1.0.0` or `v2.3.4`.

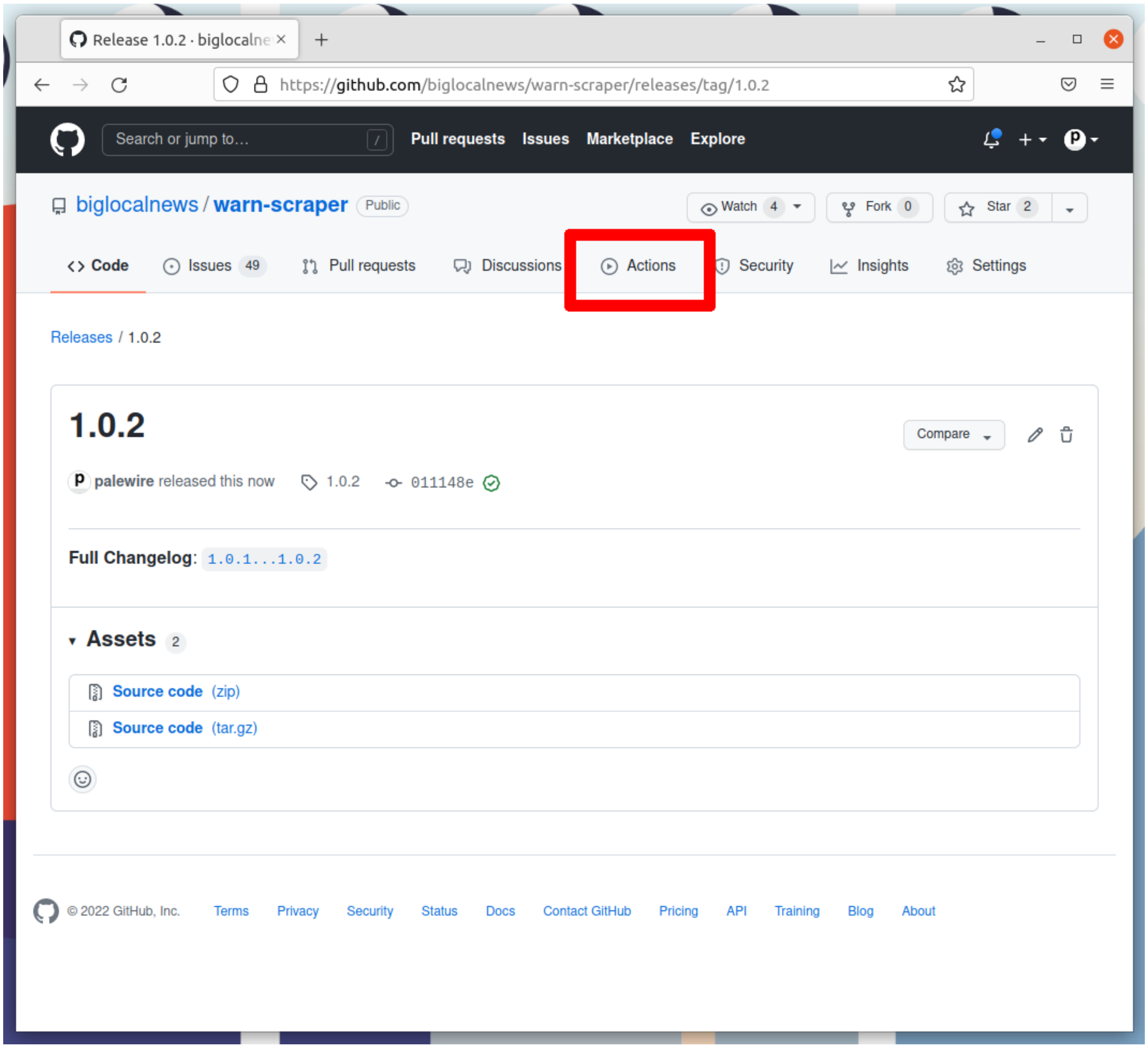
If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be `v0.2.0-alpha` or `v5.9-beta.3`.

Semantic versioning

If you're new to releasing software, we highly recommend reading about semantic versioning.

1.5.7 7. Wait for the Action to finish

GitHub will take you to a page dedicated to your new release and start an automated process that release our new version to the world. Follow its progress by clicking on the Actions tab near the top of the page.



Releases / 1.0.2

1.0.2

palewire released this now 1.0.2 011148e

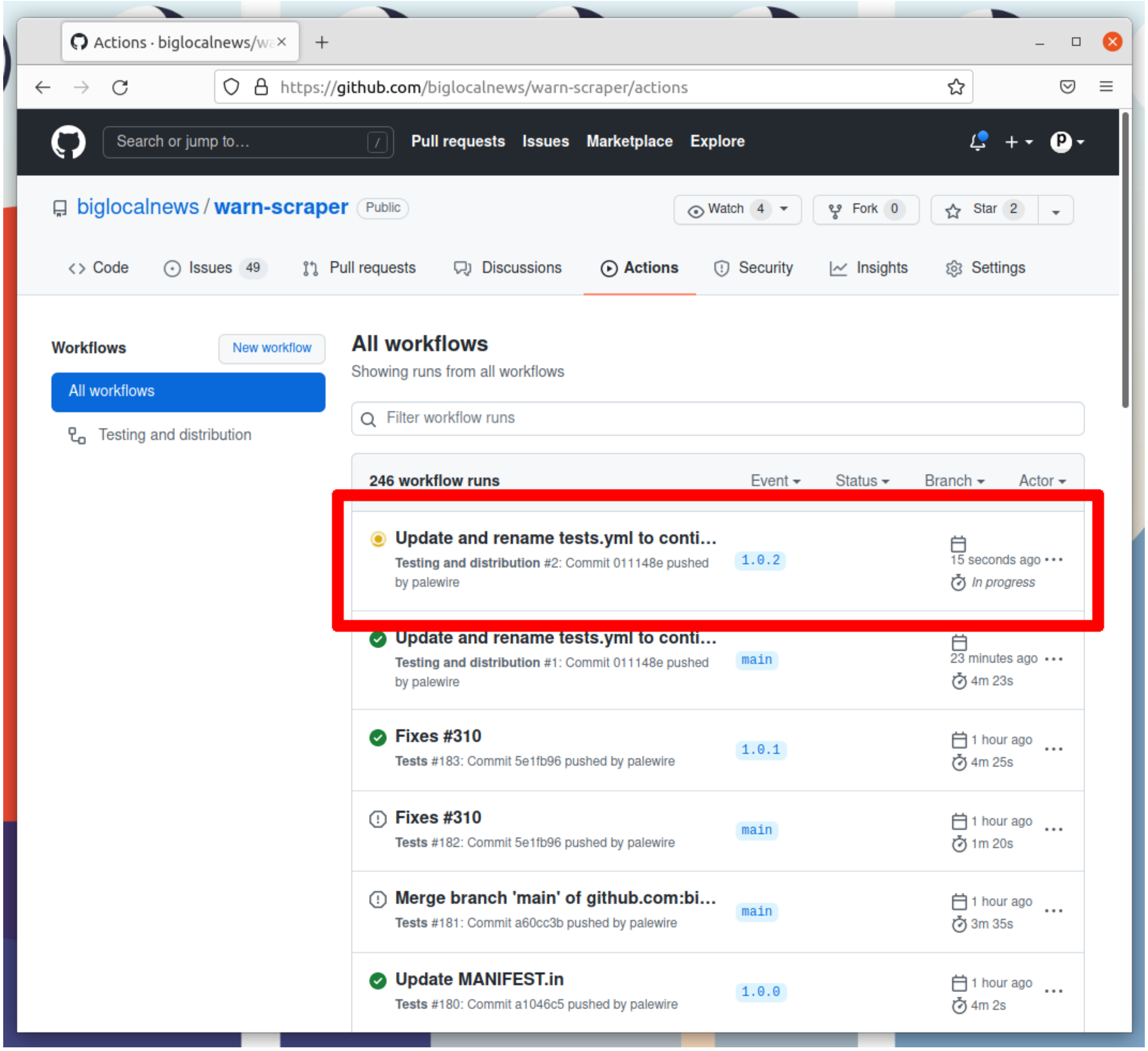
Full Changelog: `1.0.1...1.0.2`

Assets 2

Source code (zip)

Source code (tar.gz)

That will take you to the Actions monitoring page. The task charged with publishing your release should be at the top.



All workflows

Showing runs from all workflows

246 workflow runs

Run	Branch	Time
Update and rename tests.yml to conti... Testing and distribution #2: Commit 011148e pushed by palewire	1.0.2	15 seconds ago *In progress*
Update and rename tests.yml to conti... Testing and distribution #1: Commit 011148e pushed by palewire	main	23 minutes ago 4m 23s
Fixes #310 Tests #183: Commit 5e1fb96 pushed by palewire	1.0.1	1 hour ago 4m 25s
Fixes #310 Tests #182: Commit 5e1fb96 pushed by palewire	main	1 hour ago 1m 20s
Merge branch 'main' of github.com:bi... Tests #181: Commit a60cc3b pushed by palewire	main	1 hour ago 3m 35s
Update MANIFEST.in Tests #180: Commit a1046c5 pushed by palewire	1.0.0	1 hour ago 4m 2s

After a few minutes, the process there should finish and show a green check mark. When it does, visit civic-scraper's page on PyPI, where you should see the latest version displayed at the top of the page.

If the action fails, something has gone wrong with the deployment process. You can click into its debugging panel to search for the cause or ask the project maintainers for help.

1.6 Reference

1.6.1 civic_scraper

Subpackages

civic_scraper.base

Submodules

civic_scraper.base.asset

class civic_scraper.base.asset.**Asset**(*url: str*, *asset_name: Optional[str] = None*, *committee_name: Optional[str] = None*, *place: Optional[str] = None*, *place_name: Optional[str] = None*, *state_or_province: Optional[str] = None*, *asset_type: Optional[str] = None*, *meeting_date: Optional[datetime] = None*, *meeting_time: Optional[time] = None*, *meeting_id: Optional[str] = None*, *scraped_by: Optional[str] = None*, *content_type: Optional[str] = None*, *content_length: Optional[str] = None*)

Bases: `object`

Parameters

- **url** (*str*) – URL to download an asset.
- **asset_name** (*str*) – Title of an asset. Ex: City Council Regular Meeting
- **committee_name** (*str*) – Name of committee that generated the asset. Ex: City Council
- **place** (*str*) – Name of place associated with the asset. Lowercase with spaces and punctuation removed. Ex: menlopark
- **place_name** (*str*) – Human-readable place name. Ex: Menlo Park
- **state_or_province** (*str*) – Two-letter abbreviation for state or province associated with an asset. Ex: ca
- **asset_type** (*str*) – One of SUPPORTED_ASSET_TYPES. Ex: agenda
- **meeting_date** (*datetime.datetime*) – Date of meeting or None if no date given
- **meeting_time** (*datetime.time*) – Time of meeting or None
- **meeting_id** (*str*) – Unique meeting ID. For example, cominbation of scraper type, subdomain and numeric ID or date. Ex: civicplus-nc-nashcounty-05052020-382
- **scraped_by** (*str*) – civic_scraper.__version__
- **content_type** (*str*) – File type of the asset as given by HTTP headers. Ex: 'application/pdf'
- **content_length** (*str*) – Asset size in bytes

Public methods:
download: downloads an asset to a given target_path

download(*target_dir*, *session=None*)

Downloads an asset to a target directory.

Parameters
target_dir (*str*) – target directory name

Returns
Full path to downloaded file

class civic_scraper.base.asset.**AssetCollection**(*iterable=()*, */*)

Bases: `list`

to_csv(*target_dir*)

Write metadata about the asset list to a csv.

Parameters

targer_dir (*str*) – Path to directory where metadata file should be written.

Output: csv with metadata

Returns

Path to file written.

civic_scraper.base.cache

class civic_scraper.base.cache.**Cache**(*path=None*)

Bases: `object`

property artifacts_path

Path for HTML and other intermediate artifacts from scraping

property assets_path

Path for agendas, minutes and other gov file assets

property metadata_files_path

Path for metadata files related to file artifacts

write(*name*, *content*)

civic_scraper.base.constants

civic_scraper.base.site

class civic_scraper.base.site.**Site**(*base_url*, *cache=<civic_scraper.base.cache.Cache object>*, *parser_kls=None*)

Bases: `object`

Base class for all Site scrapers.

Parameters

- **base_url** (*int*) – URL to a government agency site
- **cache** (*Cache instance*) – Optional Cache instance (default: ".civic-scraper" in user home dir)
- **parser_kls** (*class*) – Optional parser class to extract data from government agency websites.

scrape(**args*, ***kwargs*) → *AssetCollection*

Scrape the site and return an AssetCollection instance.

civic_scraper.platforms

Subpackages

civic_scraper.platforms.civic_plus

Submodules

civic_scraper.platforms.civic_plus.parser

civic_scraper.platforms.civic_plus.site

Submodules

civic_scraper.runner

class `civic_scraper.runner.`**`Runner`**(*cache_path=None*)

Bases: `object`

Facade class to simplify invocation and usage of scrapers.

Arguments:

- cache_path – Path to cache location for scraped file artifact

`scrape`(*start_date*, *end_date*, *site_urls=[]*, *cache=False*, *download=False*)

Scrape file metadata and assets for a list of agency sites.

For a given scraper, scrapes file artificate metadata and downloads file artificats. Automatically generats a metadata CSV of file assets.

If requested, caches intermediate file artifacts such as HTML from scraped pages and downloads file assets such as agendas, minutes (caching and downloading are optional and are off by default).

Parameters

- **`start_date`** (*`str`*) – Start date of scrape (YYYY-MM-DD)
- **`end_date`** (*`str`*) – End date of scrape (YYYY-MM-DD)
- **`site_urls`** (*`list`*) – List of site URLs
- **`cache`** (*`bool`*) – Optionally cache intermediate file artificats such as HTML (default: False)
- **`download`** (*`bool`*) – Optionally download file assets such as agendas (default: False)

Outputs:

Metadata CSV listing file assets for given sites and params.

Returns

AssetCollection instance

exception `civic_scraper.runner.`**`ScraperError`**

Bases: `Exception`

civic_scraper.utils

civic_scraper.utils.**default_user_home**()

civic_scraper.utils.**dtz_to_dt**(*dtz*)

civic_scraper.utils.**mb_to_bytes**(*size_mb*)

civic_scraper.utils.**parse_date**(*date_str*, *format='%Y-%m-%d'*)

civic_scraper.utils.**today_local_str**()

CHAPTER

TWO

LINKS

- Documentation: http://civic-scraper.readthedocs.io/en/latest/
- GitHub: https://github.com/biglocalnews/civic-scraper
- PyPI: https://pypi.python.org/pypi/civic-scraper

PYTHON MODULE INDEX

INDEX