court-scraper

unknown

Dec 13, 2022

CONTENTS

A Python library that downloads case information from U.S. county courts

CHAPTER

ONE

DOCUMENTATION

1.1 Installation

Install the library from the Python Package Index with `pipenv`.

```
pipenv install court-scraper
```

Upon installation, you should have access to the `court-scraper` tool on the command line. Use the `--help` flag to view available sub-commands:

```
court-scraper --help
```

Note: See *the usage docs* for details on using *court-scraper* on the command line and in custom scripts.

1.1.1 Default cache directory

By default, files downloaded by the command-line tool will be saved to the `.court-scraper` folder in the user's home directory.

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

1.1.2 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 COURT_SCRAPER_DIR=/tmp/some_other_dir
```

1.1.3 Configuration

Many court sites require user credentials to log in or present CAPTCHAs that must be handled using a paid, third-party service (`court-scraper` uses Anti-captcha).

Sensitive information such as user logins and the API key for a CAPTCHA service should be stored in a YAML configuration file called `config.yaml`.

This file is expected to live inside the default storage location for scraped files, logs, etc.

On Linux/Mac, the default location is `~/.court-scraper/config.yaml`.

This configuration file must contain credentials for each location based on a Place ID, which is a snake_case combination of state and county (e.g. `ga_dekalb` for Dekalb County, GA).

Courts with a common software platform that allow sharing of credentials can inherit credentials from a single entry.

Here's an example configuration file:

```yaml
# ~/.court-scraper/config.yaml
captcha_service_api_key: 'YOUR_ANTICAPTCHA_KEY'
platforms:
  # Mark a platform user/pass for reuse in multiple sites
  odyssey_site: &ODYSSEY_SITE
    username: 'user@example.com'
    password: 'SECRET_PASS'
# Inherit platform credentials across multiple courts
ga_chatham: *ODYSSEY_SITE
ga_dekalb: *ODYSSEY_SITE
ga_fulton: *ODYSSEY_SITE

# Or simply set site-specific attributes
ny_westchester:
  username: 'user2@example.com'
  password: 'GREAT_PASSWORD'
```

CAPTCHA-protected sites

`court-scraper` uses the Anti-captcha service to handle sites protected by CAPTCHAs.

If you plan to scrape a CAPTCHA-protected site, register with the Anti-captcha service and obtain an API key.

Then, add your API key to your local court-scraper configuration file as shown below:

```yaml
# ~/.court-scraper/config.yaml
captcha_service_api_key: 'YOUR_API_KEY'
```

Once configured, you should be able to query CAPTCHA-protected sites currently supported by `court-scraper`.

1.2 Usage

court-scraper provides a command-line tool and underlying Python library that can be used to scrape data about court cases. The command-line tool supports basic search by case numbers. The Python library offers a wider range of options for more advanced use cases (e.g. filtering search results by case type).

Our project focuses on scraping data from platforms used by county-level courts. These platforms vary in features. Some only offer basic search by party or case number, whereas others support advanced search by one or more parameters such as date range and case type.

Wherever possible, *court-scraper* attempts to provide support for search by:

- **date range** - to enable automated discovery of new cases and backfilling of previous cases
- **case type** - to enable more targeted scrapes in combination with date range
- **case number** - to enable ongoing updates of open cases

The library is currently focused on acquring raw file artifacts (e.g. HTML and JSON files containing case data). *court-scraper* does not automate the extraction and standardization of data from these raw files.

Note: We hope to eventually provide tools to help with data extraction and standardization. However, due to the wide variability of case types even within a single platform, this effort remains on our long-term roadmap. We welcome *contributions* on this front!

1.2.1 Find a court to scrape

Before you can start scraping court records, you must first pinpoint a county of interest and check whether we currently support it.

Use the command-line tool's info sub-command to list currently supported counties.

If you don't see the state or county you're targeting, it's worth checking out our Issue tracker to see if it's on the roadmap. In some cases, we may be actively working on adding support for your jurisdiction. We also have a stable of scrapers that were written by others for project-specific purposes and contributed to our project for integration into our more general framework. We can provide access to these "one-off" scrapers for your customization, even if we have not yet integrated them into `court-scraper`.

1.2.2 Place IDs

court-scraper requires searches to target courts/jurisdictions in specific counties. Every jurisdiction supported by the framework has a so-called *Place ID*. These unique identifiers are in "snake case" format (i.e. lower case with underscores): `<state_postal>_<county_name>`.

For example, the *Place ID* for Tulsa, Oklahoma is `ok_tulsa`.

Whether working with the *Command line* or Custom scripts, you'll need to identify the `Place ID` for the target jurisdiction. You can use the command-line tool's info sub-command to find the `Place ID` for your jurisdiction.

1.2.3 Command line

Note: Before using the command-line tool, check out the *install docs* and read up on finding a court site to scrape.

The command-line tool helps pinpoint counties currently supported by *court-scraper* and enables scraping case files by number.

Use the `--help` flag to view available sub-commands:

```
court-scraper --help
```

Info command

The `info` sub-command lists the currently supported counties:

```
court-scraper info
```

Note: See find a site for advice if your jurisdiction is not among those listed.

Case number search

The *court-scraper* CLI's `search` sub-command is the primary way to gather case details from a county court site. You can use the tool's `--help` flag to get details on available options:

```
court-scraper search --help
```

The `search` sub-command supports scraping by case number. It requires two parameters:

- `--place-id` or `-p` - A combination of state postal and county name in "snake case" (e.g. `ok_tulsa`). The Place ID can be obtained by using the info sub-command.
- `--case-number` or `-c` - A single case number to scrape.

Here's an example search for Tulsa, Oklahoma:

```
# Scrape case details by place ID and case number
court-scraper search --place-id ok_tulsa --case-number CJ-2021-2045
```

To search for more than one case at a time, use the `--case-numbers-file` (or `-f`) flag with a text file containing case numbers on separate lines.

For example, if you create a *case_numbers.txt* file with the below case numbers:

```
# case_numbers.txt
CJ-2021-2045
CJ-2021-2046
```

You can then search using the *case_numbers.txt* file:

```
court-scraper search --place-id ok_tulsa --case-numbers-file case_numbers.txt
```

Browser mode

Scrapers that use Selenium to drive a web browser by default run in "headless" mode (i.e. the browser will not run visibly). In order to run a Selenium-based scraper with the browser, which can be helpful for debugging, use the `--with-browser` flag:

```
court-scraper search --with-browser --place-id wi_green_lake --case-number 2021CV000055
```

File storage

Files scraped by the `search` sub-command are saved to a standard – but configurable – location in the user's home directory, based on the court's Place ID (`~/.court-scraper/cache/<place_id>` on Linux/Mac).

For example, HTML files scraped for Tulsa, Oklahoma are stored in `~/.court-scraper/cache/ok_tulsa`.

Metadata db

The `search` sub-command stores basic metadata about scraped cases in a SQLite database located in the standard cache directory: `~/.court-scraper/cases.db`.

The database can be helpful for quickly checking which cases have been scraped.

It stores the following fields:

- `created` (*datetime*) - The date and time of the case was initially scraped.
- `updated` (*datetime*) - The date and time of last scrape for the case.
- `place_id` (*str*) - The state postal and county name in "snake case" (e.g. *ok_tulsa*).
- `number` (*str*) - The case number.
- `filing_date` (*date*) - The filing date of the case (if available).
- `status` (*str*) - Case status (if available).

1.2.4 Custom scripts

court-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 more advanced scenarios such as searching by date and filtering by case type.

Note: Court websites offer different search functionality, so it's important to review the site and its corresponding Site class (and search methods) in this library to get a sense of supported features.

Scrape case details by number

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

Create an instance of `Site` by passing it the Place ID for the jurisdiction. Then call the `search` method with one or more case numbers:

```
from court_scraper import Site
site = Site('ok_tulsa')
case_numbers=['CJ-2021-1904', 'CJ-2021-1905']
results = site.search(case_numbers=case_numbers)
```

Note: `Site` provides a generic interface to simplify import and configuration of platform-specific Site classes, such as `court_scraper.platforms.oscn.site.Site`. Platform Site classes typically have varying options for initialization and search, so it's a good idea to review their options when using this generic Site class.

Scrape by date

Some court sites support date-based search. In such cases, you can use the platform's `Site.search_by_date` method to scrape data for one or more days.

By default, `search_by_date` only gathers case metadata (e.g. case numbers, filing dates, status, etc.) that typically appear on a results page after performing a search.

Note: See below for details on scraping case detail file artifacts (e.g. HTML, JSON, etc.).

To scrape case metadata for the current day:

```
from court_scraper import Site
site = Site('ok_tulsa')
results = site.search_by_date()
```

To search a range of dates, use the `start_date` and `end_date` arguments. Their values must be strings of the form `YYYY-MM-DD`. The below code scrapes metadata for cases filed in Tulsa, Oklahoma during January 2021:

```
from court_scraper import Site
site = Site('ok_tulsa')
results = site.search_by_date(start_date='2021-01-01', end_date='2021-01-31')
```

Scrape case details

Court sites typically provide more detailed case information on separate pages devoted to a case. Depending on the site, these pages can include:

- Case type
- Case status
- Litigant information (i.e. names and addresses)
- Judge name(s)

- Events related to the case (e.g. filings and decisions)

Links to case detail pages are listed on a results page after conducting a search. These are typically HTML, but may be JSON or other file formats depending on the site.

By default, `search_by_date` only scrapes metadata from search results pages (as described in Scrape by date).

To scrape case detail files, pass the `case_details=True` keyword argument:

```python
from court_scraper import Site
site = Site('ok_tulsa')
results = site.search_by_date(
    start_date='2021-01-01',
    end_date='2021-01-31',
    case_details=True # Fetches case detail files
)
```

Filter by case type

Some court sites support a variety of parameters for more targeted filtering of search results. These filters can be useful for more surgical scrapes, and in scenarios where a site truncates results. If a site limits search results to 500 records, for example, scraping in a more targeted way with filters can help stay under that cap.

To determine if a site supports case-type filtering, you should review the court's website and the corresponding `Site` class in *court-scraper*.

For example, the Wisconsin court system's Advanced Search page offers a variety of additional search parameters. In *court-scraper*, the site's corresponding `search_by_date` method supports a `case_types` argument that accepts a list of one or more case types.

Note: For Wisconsin, these case types are two-letter, upper-case codes that can be found by examining the source code for the *Case types* select menu on the Advanced Search page.

Here's a sample usage that searches for civil (CV) and small claims (SC) cases on July 1, 2021 in Milwaukee, WI:

```python
from court_scraper import Site
site = Site('wi_milwaukee')
results = site.search_by_date(
    start_date='2021-07-01',
    end_date='2021-07-01',
    case_types=['CV', 'SC'] # Civil and Small Claims case types
)
```

1.3 Contributing

Contributions are welcome and appreciated! Every little bit helps, and credit will always be given. There are plenty of ways to get involved besides writing code. We've listed a few options below.

1.3.1 Ways to Contribute

Write a scraper

Don't see a scraper for your state or county? We'd love to have you write a scraper to help us expand coverage!

Our Writing a scraper page is the best place to get started. It's also a good idea to file a new ticket for the work on our Issue tracker, or claim the ticket if one already exists. We're happy to talk through strategies for scraping and integration with the framework, so please do reach out!

Report Bugs

Report bugs on our Issue tracker.

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 Issue tracker for bugs. Anything tagged with "bug" 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

We could always use more documentation, whether as part of the official court-scraper docs, in docstrings, or even on the web in blog posts, articles, and such. Our official docs use Markdown and Sphinx. You can find the files in the repository's `docs` folder.

1.3.2 Get Started!

Ready to contribute? Check out our docs on Writing a scraper and *Testing*, and see below steps on setting up `court-scraper` for local development.

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

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

3. Set up a local virtual environment and install dev dependencies for local development with Pipenv:

```
cd court-scraper/
pipenv install --dev
```

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:

```
# Lint and test for current Python version
make test
make lint
# If you're working on documentation, you can build and preview the docs with
make docs
make servedocs
```

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.

1.3.3 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 the Actions tab on GitHub and make sure that the tests pass for all supported Python versions.

1.4 Writing a scraper

court-scraper's main goal is to serve as a framework for acquiring basic court case metadata and raw file artifacts (e.g. HTML, JSON, PDFs, etc.) for county-level courts.

These files can then be further processed in separate scripts that perform data extraction and standardization to support the needs of a given project.

We're especially focused on platforms used by a large number of county court sites, although we expect to create "one-off" scrapers for bespoke sites when necessary.

We also anticipate situations where county-level data simply isn't available online. Such cases will require requesting data on a regular basis (and possibly paying for it).

If you're thinking of scraping a court site, it's important to conduct some research to determine what data a court site provides. Some jurisdictions provide a simple search with case details readily accessible online (and easily scrapable). Others provide multiple ways of accessing case data, such as a free and open site that allows searching for case metadata, while hiding more detailed case information behind logins, CAPTCHAs, paywalls, or some combination of these barriers.

Additionally, sites typically include legal restrictions on access and use of data.

When embarking on scraping a court site, it's important to understand their offerings and the legal restrictions surrounding use of the site and its data. Please perform your due diligence and reach out to discuss a particular site if you have questions about strategy!

1.4.1 Devise a scraping strategy

Before coding a new scraper, take some time to determine the best scraping strategy by interacting with and dissecting the site.

Whenever possible, we favor scrapers that gather data using basic HTTP GET or POST calls using the Python requests library. Sites that are heavy on dynamically generated content and pose other challenges may require browser automation via Selenium. Such cases are unavoidable, although often you may find it's possible to use a combination of both scraping strategies to optimize the speed of the scraper. For example, the `Wisconsin scraper` uses both libraries to achieve faster scrapes while handling (and minimizing the cost) of CAPTCHAs.

1.4.2 Code a scraper

Add a Site class

Note: Check out the docs for getting started on code contributions for details on setting up a fork for local development.

The main task involved in contributing a scraper is creating a `Site` class that provides a `search` method capable of scraping one or more case numbers.

For courts that offer date-based search, `Site` should also have a `search_by_date` method. If the date search can be filtered by one or more case types, the method should include support for this filter as well.

Lastly, sites that require login should have a `login` method.

These methods should have standard signatures in order to support automated scraping and for integration with *court-scraper*'s *commnand-line tool*.

Below is a simplified example of a scraper for an imaginary platform called *Court Whiz*. Each method notes its expectations, and we use type annotations to signal expected return values:

```python
# court_scraper/platforms/court_whiz/site.py

from typing import List
from court_scraper.case_info import CaseInfo

class Site:

    def __init__(self, place_id):
        self.place_id = place_id
        self.url = "https://court-whiz.com"

    def search(self, case_numbers=[]) -> List[CaseInfo]:
        # Perform a place-specific search (using self.place_id)
        # for one or more case numbers.
        # Return a list of CaseInfo instances containing case metadata and,
        # if available, HTML for case detail page
        pass
```

(continues on next page)

(continued from previous page)

```
    def search_by_date(self, start_date=None, end_date=None, case_details=False, case_
↪types=[]) -> List[CaseInfo]:
        # Perform a place-specific, date-based search.
        # Defaut to current day if start_date and end_date not supplied.
        # Only scrape case metadata from search results pages by default.
        # If case_details set to True, scrape detailed case info
        # Apply case type filter if supported by site.
        # Return a list of CaseInfo instances
        pass

    def login(self, username, password):
        # Perform login with username and password
        pass
```

Site classes for scrapers specific to a single county should live in the `court_scraper.scrapers` namespace under a package based on the jurisdiction's Place ID.

For example, the Site class for Westchester County would live in `court_scraper.scrapers.ny_westchester.site.py`.

Many counties use common software platforms, such as Odyssey by Tyler Technologies, to provide case information.

To add a platform-based scraper for use in more than one jurisdiction, add a site class to the `court_scraper.platforms` namespace. For example, `court_scraper.platforms.odyssey.site.Site`.

Note: We've provided some base classes and helper functions to help with common scenarios (e.g. see `SeleniumHelpers` and functions in `court_scraper.utils`).

Add tests

New site classes should include test coverage for the `search` and `search_by_date` methods.

Check out our *Testing docs* and review test modules for the Odyssey, Oklahoma (oscn) or Wisconsin (wicourts) site classes for examples that can help you get started.

Update *court_scraper.site.Site*

The `court_scraper.site.Site` class provides a simpler interface for looking up and working with a jurisdiction's Site class.

If your new Site class has some initialization needs beyond simply providing a Place ID, you may need to update `court_scraper.site.Site` with special handling for your new Site class.

Even if you don't update `court_scraper.site.Site`, it's a good idea to add at least one high-level integration test in `tests/test_site.py` for your new Site class to ensure it's handled correctly.

1.4.3 CLI Integration

Integration with *court-scraper*'s *command-line tool* requires several steps, as detailed below.

Create a Runner

First, you must create a `Runner` class capable of driving the newly implemented `Site` class. Runners generally perform the following taks:

- Instantiate the `Site` class
- Call `Site.search` with values passed in by `court_scraper.cli`
- Set sensible defaults, as needed
- Perform caching of scraped file artifacts
- Log information to the command-line, as needed

See the runners for `Oklahoma` or `Odyssey` for reference implementations.

Sites Meta CSV

In order for our CLI tool to execute scrapes for a given jurisdiction, the jurisdiction must be added to sites_meta.csv. This file contains the following fields:

- `state` - 2-letter state abbreviation, lower cased
- `county` - lower-case name of county (without the word "County")
- `site_type` - Base name of the Python package where the Site class lives (e.g. `odyssey` or `wicourts`)
- `site_version` - Platform based sites may have multiple versions. Use this field to denote a new version of a platform-based site.
- `captcha_service_required` - Mark as True if a site presents CAPTCHAs
- `home_url` - Starting page for a platform used by many jurisdictions at separate domains (e.g. `odyssey`)

It's important to note that *every jurisdiction covered* by a scraper must be entered in sites_meta.csv, even if the sites share a common platform.

For example, there are separate entries in sites_meta.csv for most counties in Washington State. These jurisdictions use the Odyssey platform, but they live at different domains. sites_meta.csv provides a single place to store the home URL and other metadata for each of these counties.

1.5 Discovering court websites

Author: Amy DiPierro Version: 2020-09-08

This file describes further resources for finding other court websites, with an emphasis on Tyler Technologies' Odyssey sites.

1.5.1 Finding Odyssey subdomains

Strategy 1: nmmapper.com

- This subdomain finder is the best tool I've found to search for subdomains.
- Since Odyssey websites don't always have a predictable subdomain, it will be good ot continue to search for new subdomains as we come across them.
- Here are some searches to run:
  - tylerhost.net/
  - tylerhost.net/Portal/

Strategy 2: Google Custom Search API

- We can use Google's Custom Search API to run targetted searches that surface websites built with Odyssey. I've not run this yet but it might be worth it.
- Some suggested searches that turn up promising results:
  - court portal "© 2020 Tyler Technologies, Inc." -site:tylertech.com -iTax -stock -taxes

1.5.2 Existing efforts to scrape court data

- The **Police Data Accessibility Project**, an open data initiative started on Reddit this summer, has already compiled some basic databases of court websites upon which we can build:
  - The group's Public Access to Court Records State Links.csv GoogleSheet contains a partial list of court websites with the names of vendors sometimes noted.
  - It might be worth glancing at their GitHub and Slack channel from time to time to see if there are opportunities to learn from their research and code.
- The **Tubman Project**, a nonprofit that is trying to build software to "make legal defense available to the masses", has also compiled some data on Tyler Technologies platforms in this thread.

1.6 Testing

court-scraper is developed primarily on Python 3.7 and uses the pytest library for unit testing. We use pytest-vcr for scrapers that use the `requests` library (e.g. `oscn.Site`). Scrapers that use Selenium should include a minimal set of live webtests to ensure correct functionality and guard against regressions.

Note: Selenium and other long-running tests should be marked as slow to enable optional running of these tests.

1.6.1 Install and run tests

Assuming you've cloned this repo locally and installed test and application dependencies, you can run tests by executing pytest in an active virtual environment:

```
cd court-scraper/
pipenv install --dev

# Execute tests
pipenv run pytest
```

1.6.2 Slow tests

Slow-running tests should be marked as such:

```
@pytest.mark.slow
def test_something_slow():
    ...
```

Slow tests are skipped by default. To run them, pass the `--runslow` flag when invoking pytest:

```
pytest --runslow
```

1.6.3 Live tests

Tests that hit live web sites should be marked as `webtest`, allowing them to be executed selectively:

```
@pytest.mark.webtest
def test_that_hits_live_website():
    ...

# On the command line, run only tests marked as "webtest"
pytest -m webtest
```

In many cases, tests that hit live websites should be marked as both `webtest` and `slow`:

```
@pytest.mark.webtest
@pytest.mark.slow
def test_that_hits_live_website():
    ...

# On the command line, use both flags to target long-running webtests
pytest --runslow -m webtest
```

Live web tests of Selenium-based scrapers will open a web browser by default. All tests of Selenium scrapers should use the `headless` fixture in order to provide the ability to disable running tests in browser.

These tests should typically be marked as `slow` and `webtest` as well.:

```
@pytest.mark.webtest
@pytest.mark.slow
def test_selenium_scrape(headless):
```

You can activate headless mode when running pytest by using the `--headless` flag:

```
pytest --headless --runslow
```

1.6.4 Test login credentials

Tests that hit live web sites may require authentication, as in the case of some Odyssey sites such as Dekalb and Chatham counties in Georgia.

Such tests require creating user accounts and adding login credentials to a local YAML *configuration* file.

1.7 Releasing

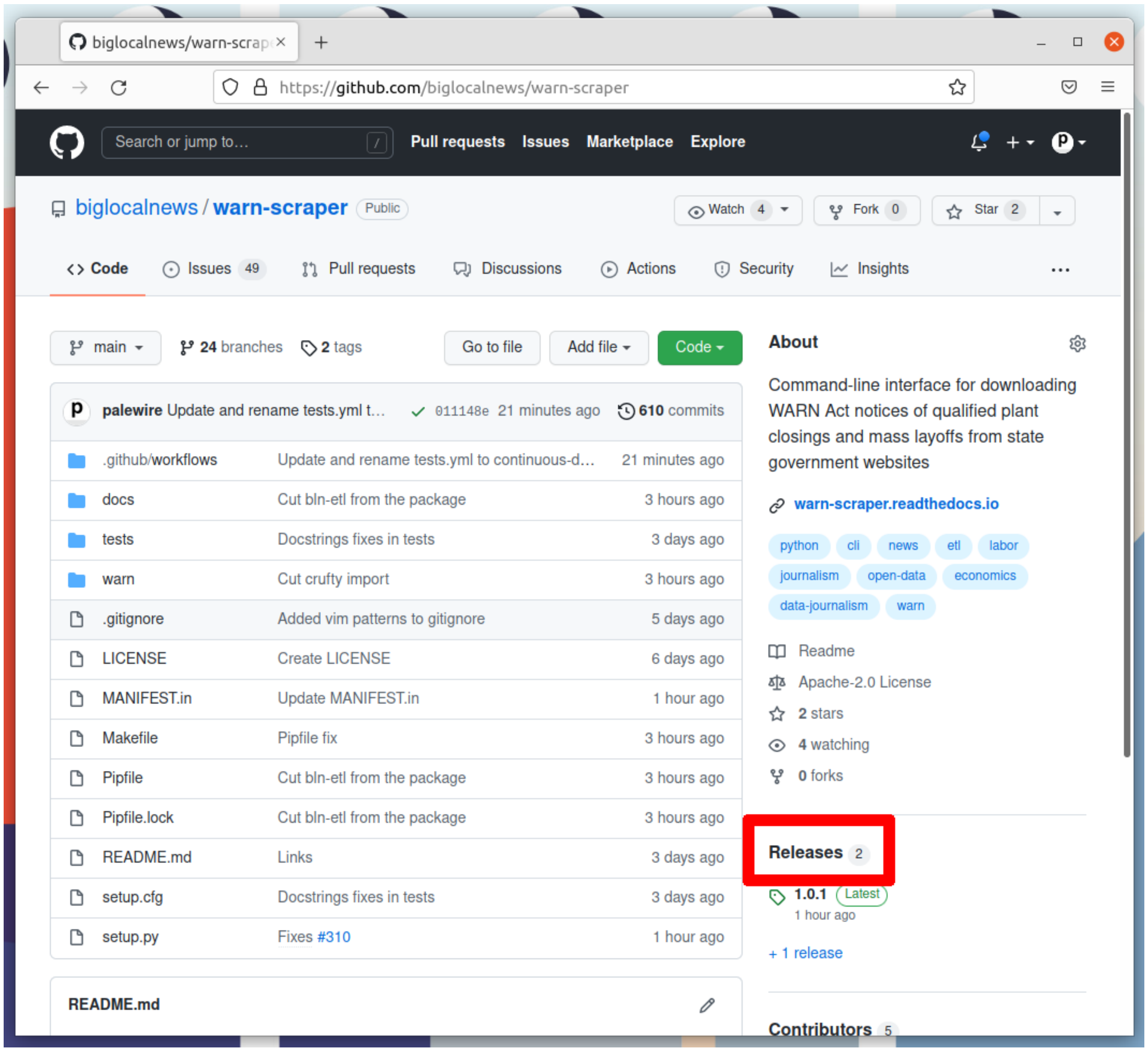
Our release process is automated as a continuous deployment via the GitHub Actions framework. The logic that governs the process is stored in the `workflows` directory.

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/court-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.

1.7.1 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

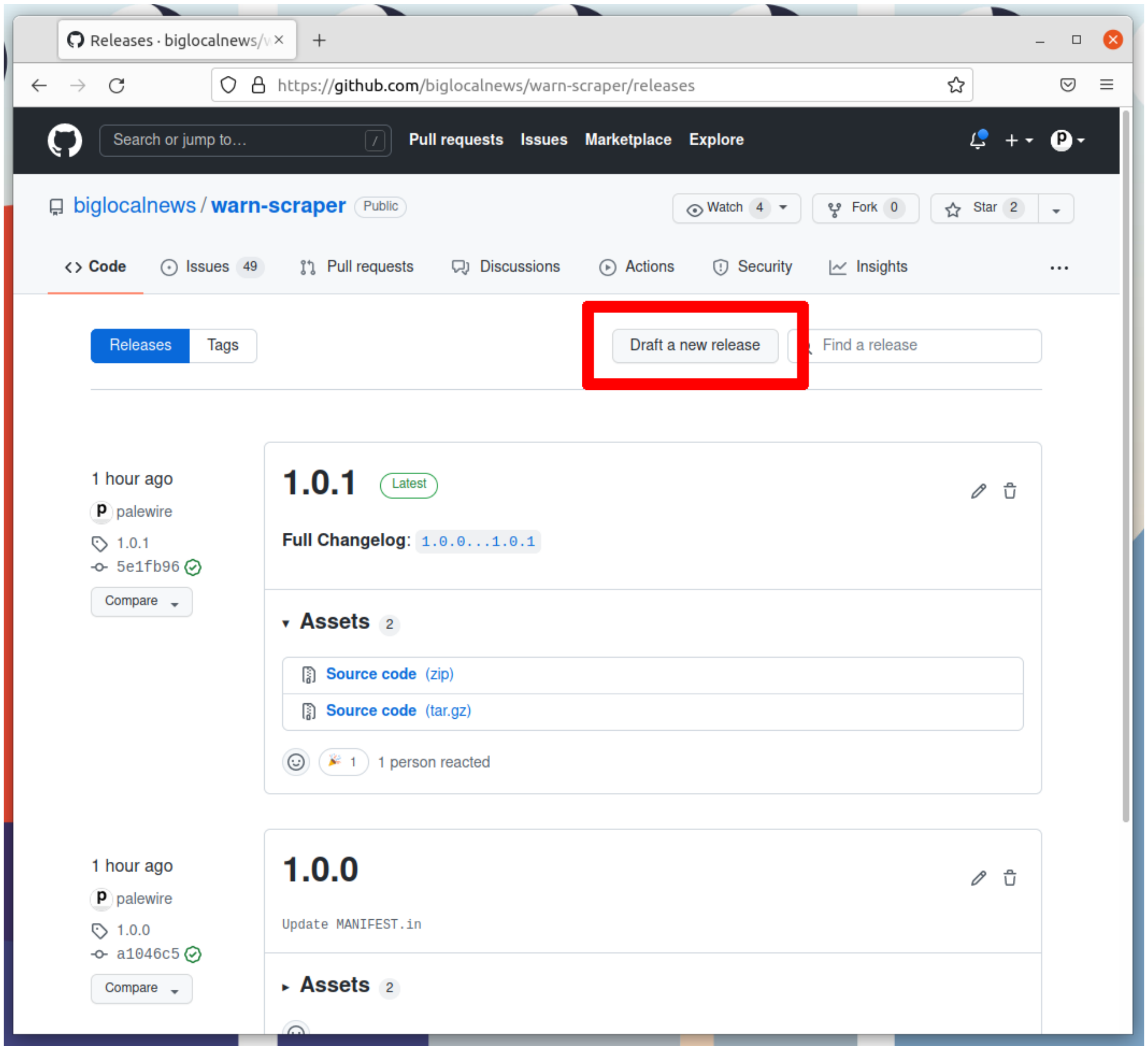
Readme
Apache-2.0 License
2 stars
4 watching
0 forks

Releases 2
1.0.1 Latest
1 hour ago
+ 1 release

Contributors 5

1.7.2 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.



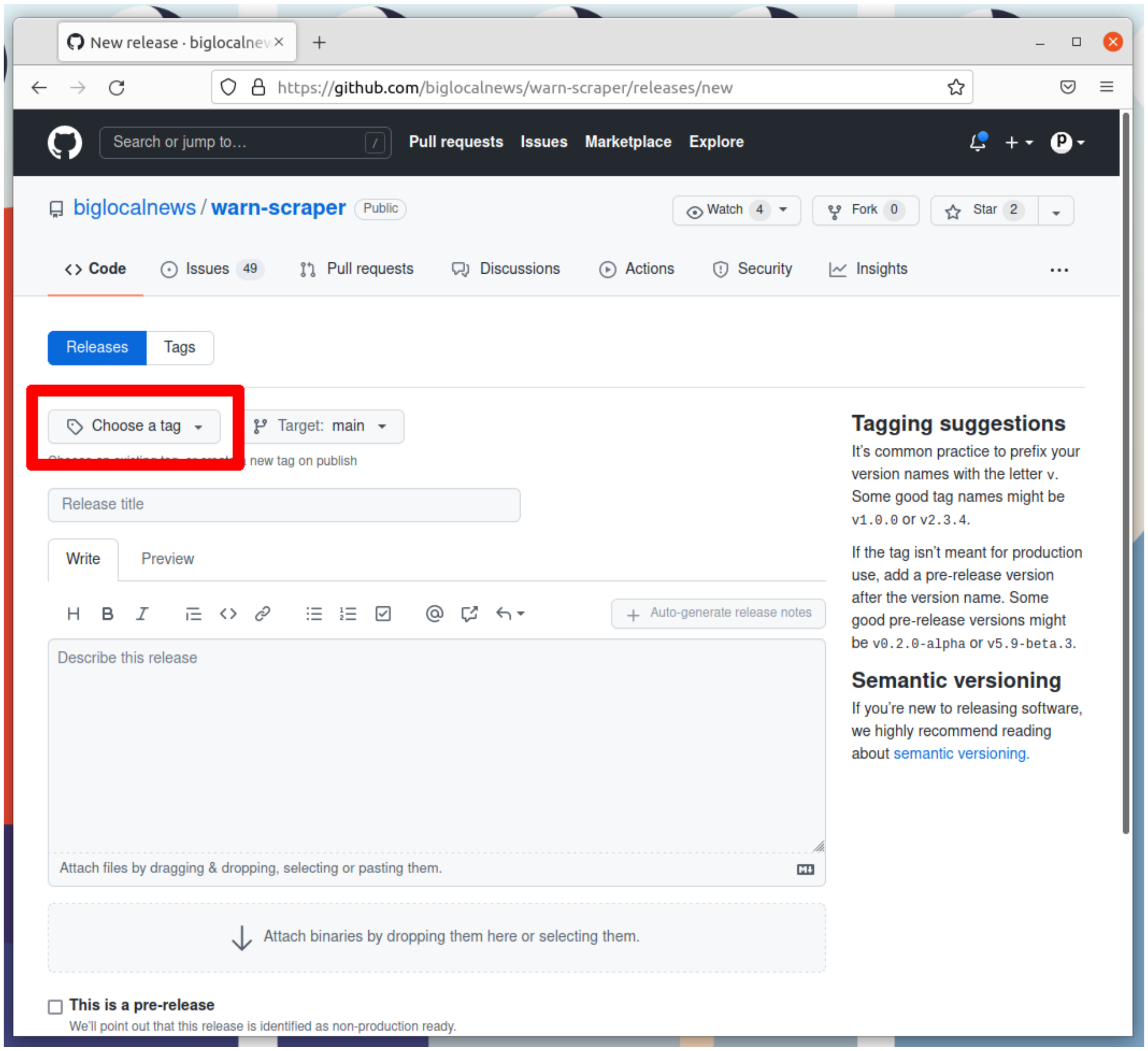
1.7.3 3. Create a new tag

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 · biglocalnews
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

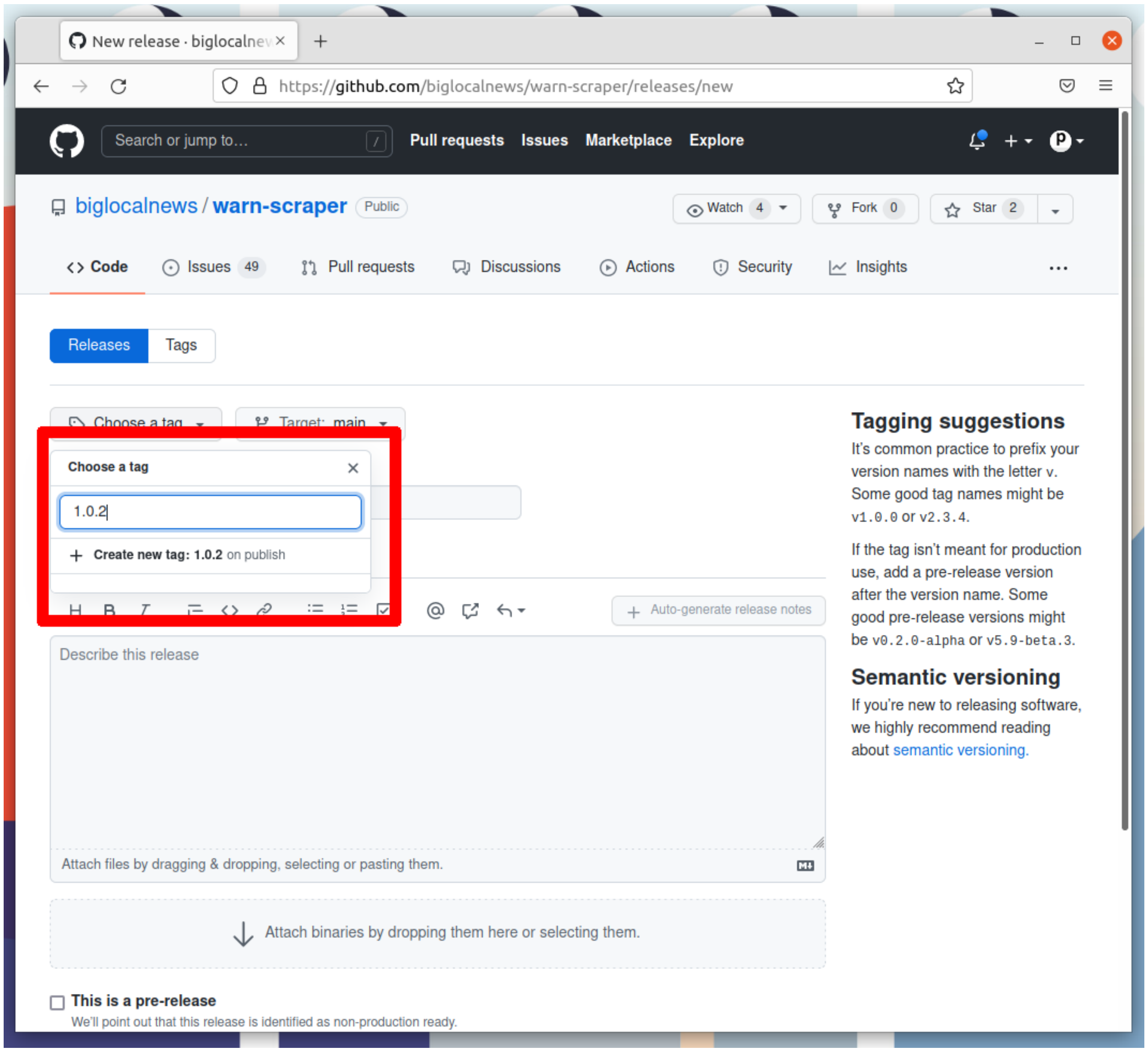
It's common practice to prefix your version names with the letter v. Some good tag names might be `v1.0.0` or `v2.3.4`.

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.


New release · biglocalnews
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
Choose a tag
1.0.2
Create new tag: 1.0.2 on publish
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`.

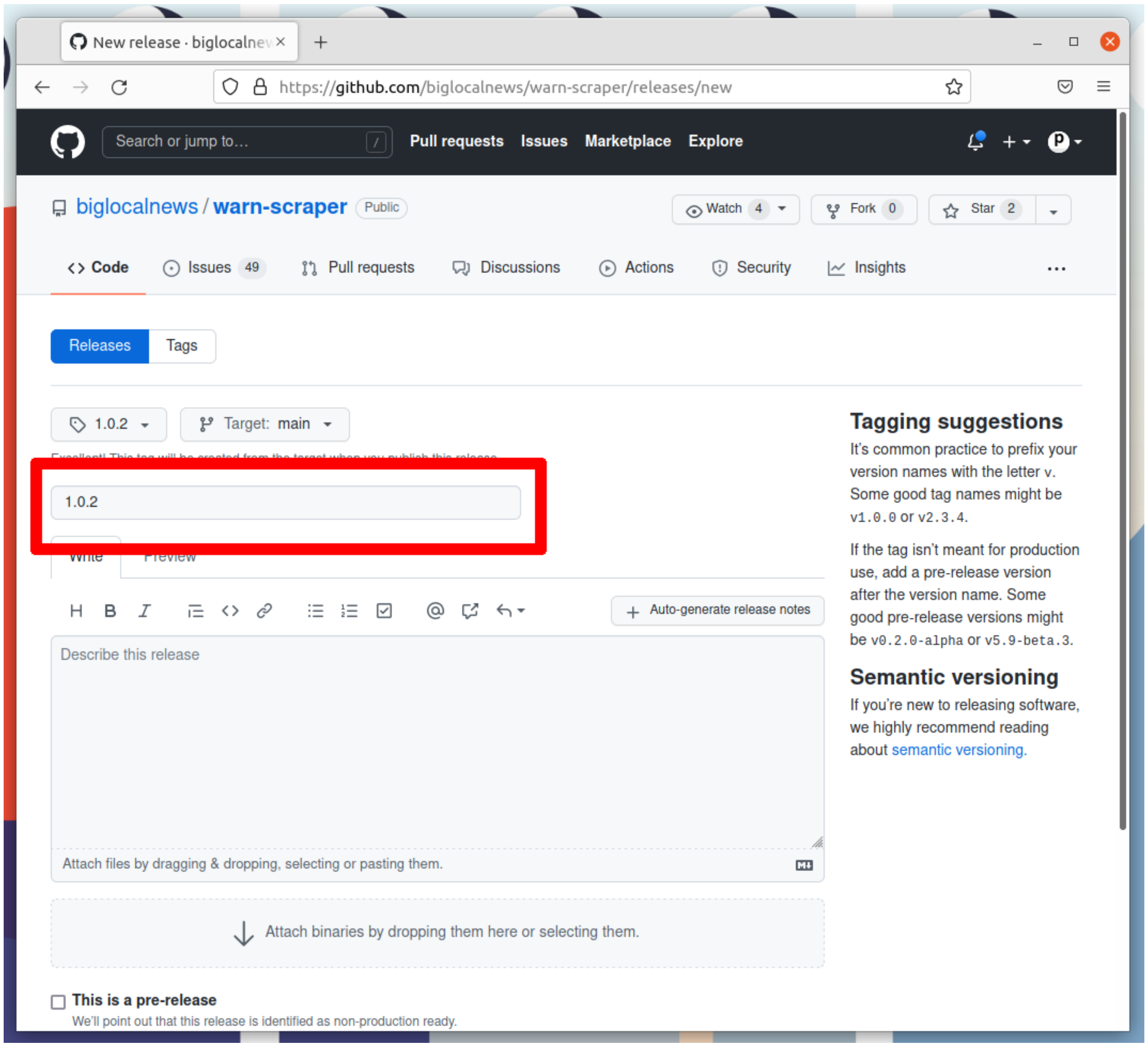
If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be `v0.2.0-alpha` or `v5.9-beta.3`.

Semantic versioning

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

1.7.4 4. Name the release

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


New release · biglocalnews
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.
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.

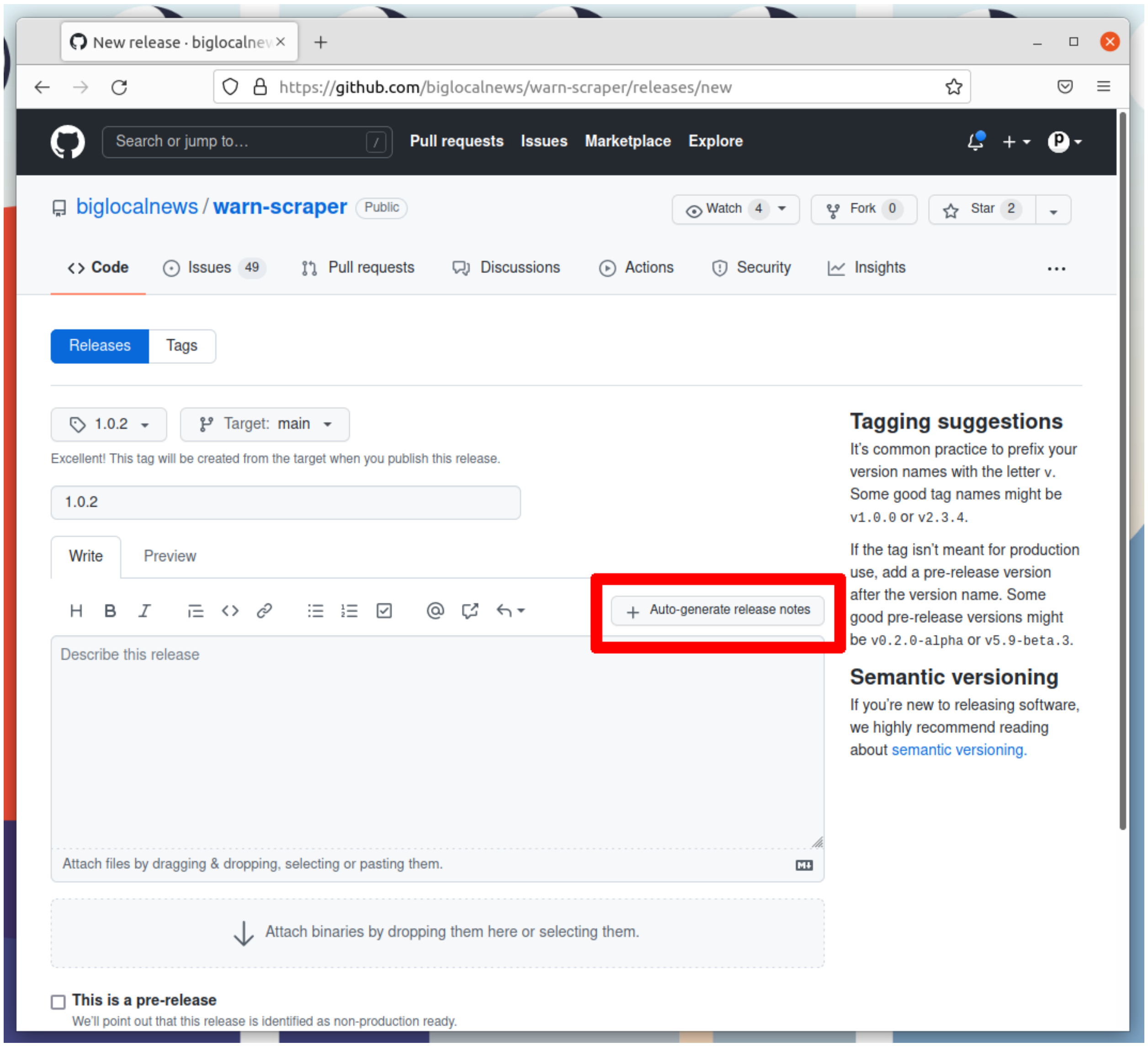
If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be v0.2.0-alpha or v5.9-beta.3.

Semantic versioning

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

1.7.5 5. Auto-generate release notes

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



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

1.0.2

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

Attach binaries by dropping them here or selecting them.

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

Tagging suggestions

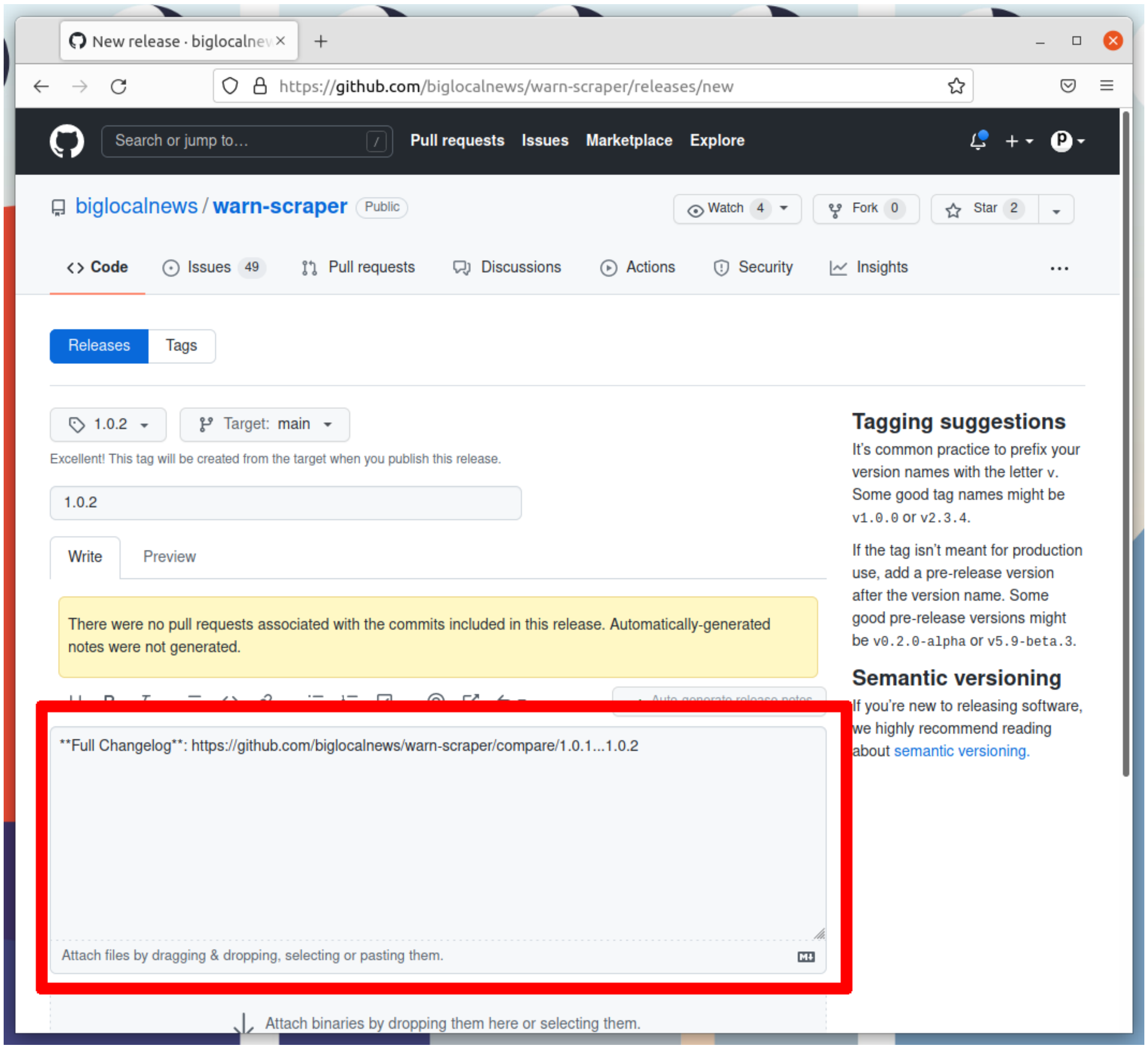
It's common practice to prefix your version names with the letter v. Some good tag names might be `v1.0.0` or `v2.3.4`.

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.

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



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.

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.

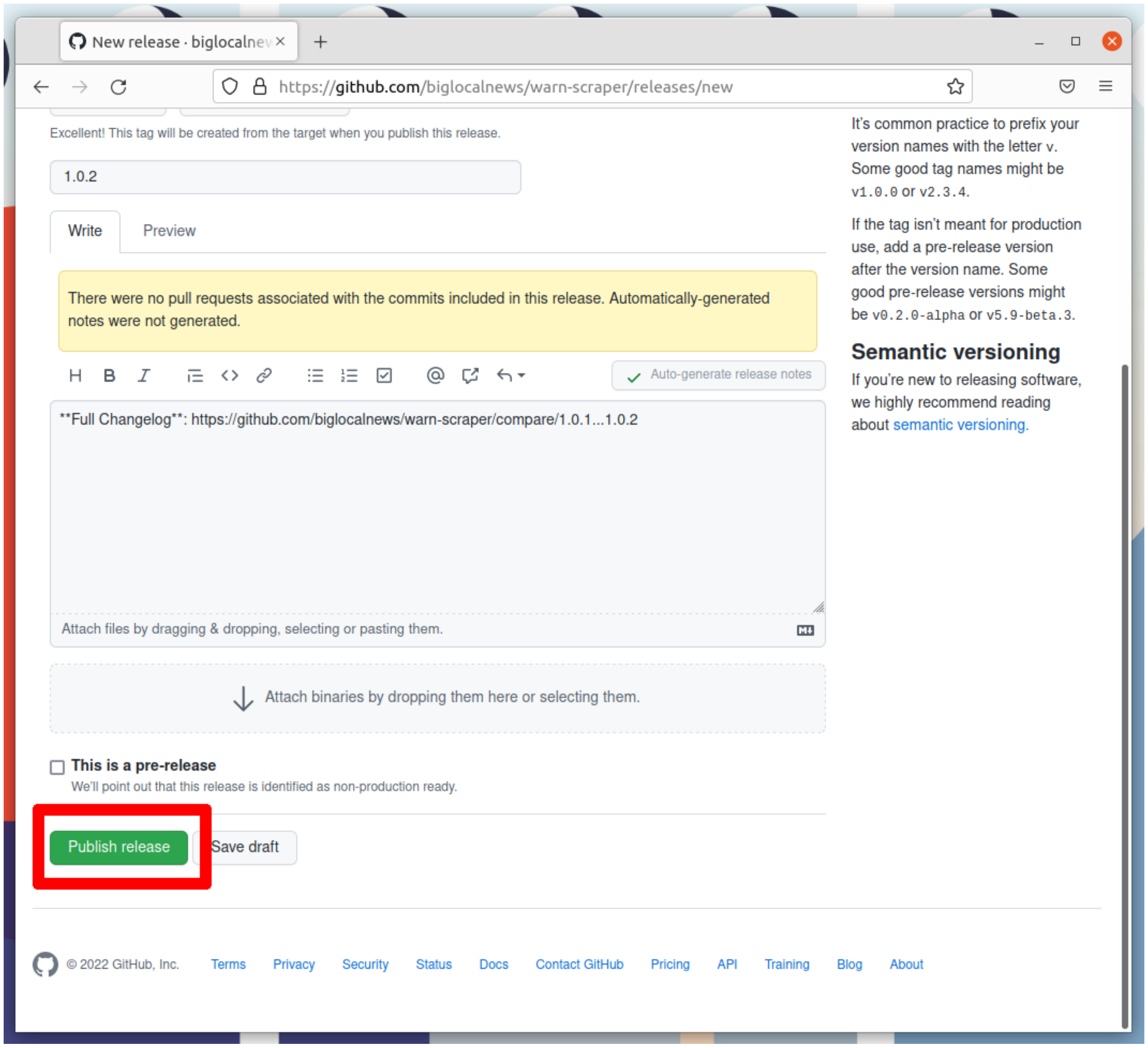
If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be v0.2.0-alpha or v5.9-beta.3.

Semantic versioning

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

1.7.6 6. Publish the release

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


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

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.

Auto-generate release notes

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.

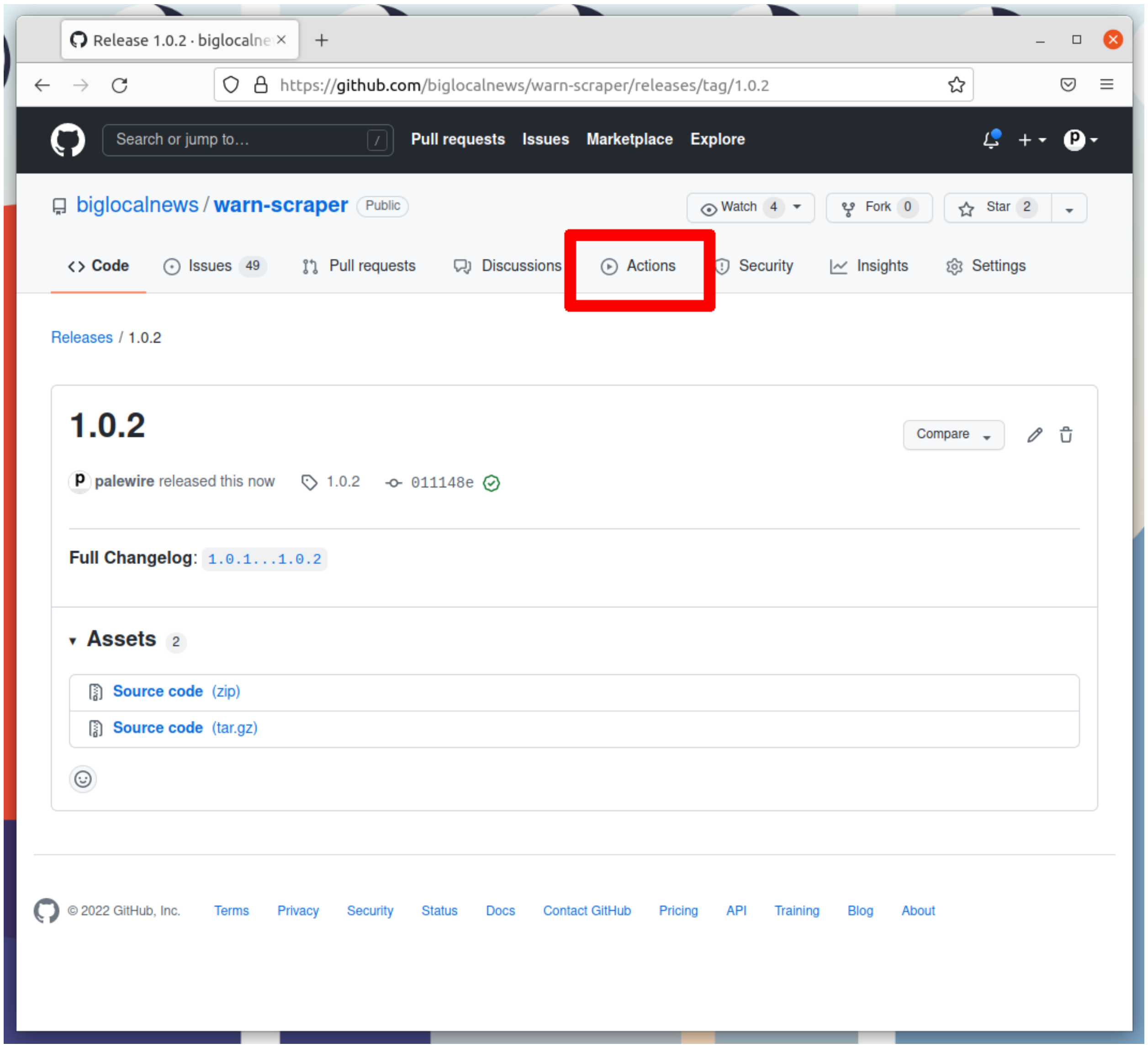
If the tag isn't meant for production use, add a pre-release version after the version name. Some good pre-release versions might be v0.2.0-alpha or v5.9-beta.3.

Semantic versioning

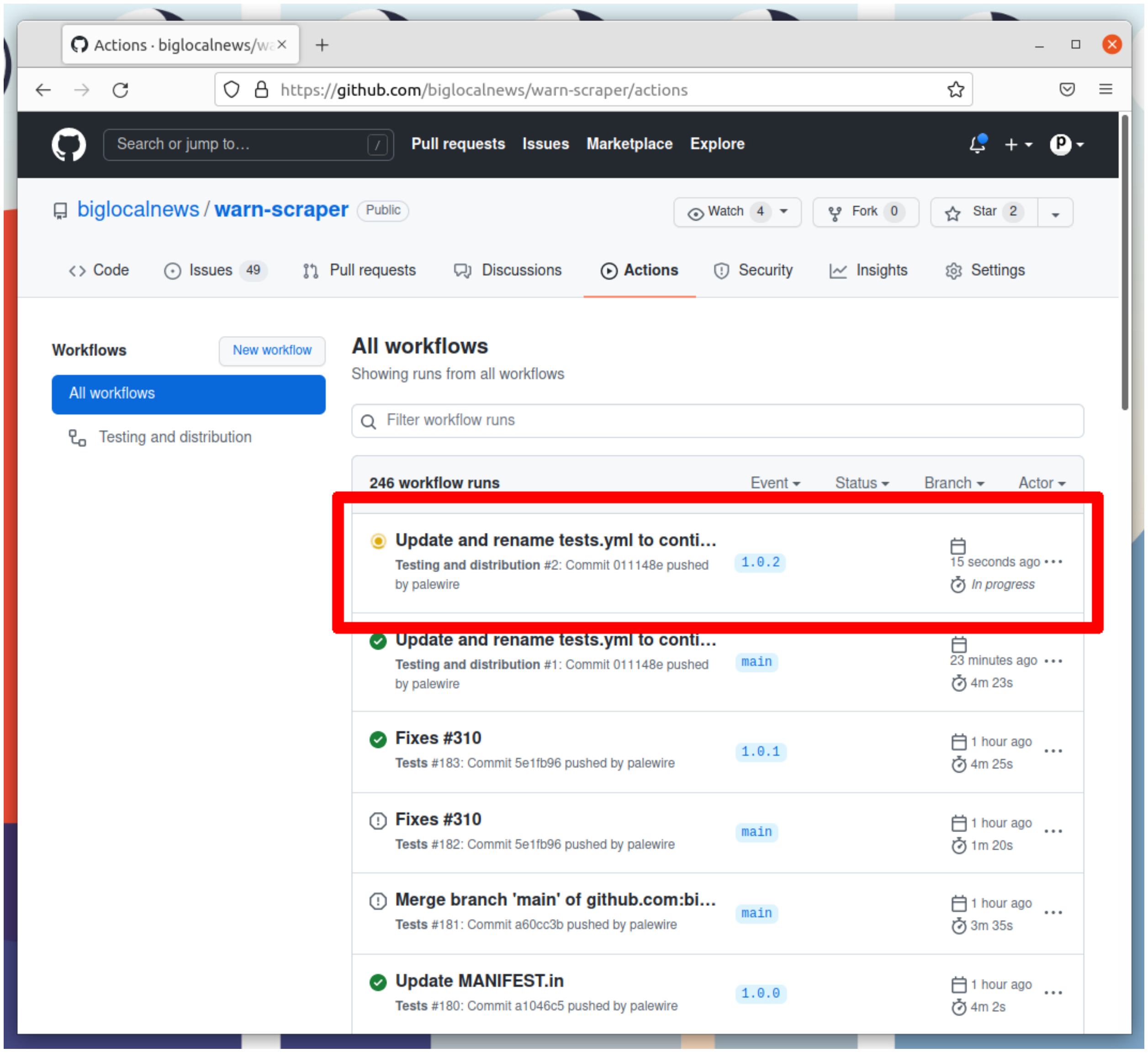
If you're new to releasing software, we highly recommend reading about semantic versioning.

1.7.7 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.



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



Actions · biglocalnews/w

https://github.com/biglocalnews/warn-scraper/actions

biglocalnews / warn-scraper Public

All workflows

Showing runs from all workflows

246 workflow runs

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 court-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.

CHAPTER TWO

LINKS

- Gitter for chat: gitter.im/court-scraper/general
- Documentation: court-scraper.readthedocs.io
- GitHub: github.com/biglocalnews/court-scraper
- PyPI: pypi.org/project/court-scraper/