Using a self-rewriting README powered by GitHub Actions to track TILs

I've started tracking TILs—Today I Learneds—inspired by this five-year-and-counting collection by Josh Branchaud on GitHub (found via Hacker News). I'm keeping mine in GitHub too, and using GitHub Actions to automatically generate an index page README in the repository and a SQLite-backed search engine.

TILs

Josh describes his TILs like this:

> A collection of concise write-ups on small things I learn day to day across a variety of languages and technologies. These are things that don't really warrant a full blog post.

This really resonated with me. I have five main places for writing at the moment.

- This blog, for long-form content and weeknotes.
- Twitter, for tweets—though Twitter threads are tending towards a long-form medium these days. My Spider-Verse behind-the-scenes thread ran for nearly a year!
- My blogmarks—links plus short form commentary.
- Niche Museums—effectively a blog about visits to tiny museums. It's on hiatus during the pandemic though.
- GitHub issues. I've formed the habit of thinking out loud in issues, replying to myself with comments as I figure things out.

What's missing is exactly what TILs provide: somewhere to dump a couple of paragraphs about a new trick I've learned, with chronological order being less important than just getting them written down somewhere.

I've intermittently used gists for things like this in the past, but having them in an organized repo feels like a much less ad-hoc solution.

So I've started my own collection of TILs in my simonw/til GitHub repository.

Automating the README index page with GitHub Actions

The biggest feature I miss from reStructuredText when I'm working in Markdown is automatic tables of content.

For my TILs I wanted the index page on GitHub to display all of them. But I didn't want to have to update that page by hand every time I added one—especially since I'll often be creating them through the GitHub web interface which doesn't support editing multiple files in a single commit.

I've been getting a lot done with GitHub Actions recently. This felt like an opportunity to put them to more use.

So I wrote a GitHub Actions workflow that automatically updates the README page in the repo every time a new TIL markdown file is added or updated!

Here's an outline of how it works:

- It runs on pushes to the master branch (no-one else can trigger it by sending me a pull request). It ignores commits that include the `README.md` file itself—otherwise commits to that file made by the workflow could

trigger further runs of the same workflow. UPDATE: Apparently this isn't necessary.

- It checks out the full repo history using actions/checkout@v2 with the `fetch-depth: 0` option. This is needed because my script derives created/updated dates for each TIL by inspecting the git history. I learned a few days ago that this mechanism breaks if you only do a shallow check-out of the most recent commit!
- It sets up Python, configures pip caching and installs dependencies from my requirements.txt.
- It runs my build_database.py script, which uses GitPython to scan for all `*/*.md` files and find their created and updated dates, then uses sqlite-utils to write the results to a SQLite database on the GitHub Actions temporary disk.
- It runs update_readme.py which reads from that SQLite database and uses it to generate the markdown index section for the README. Then it opens the README and replaces the section between the `<!-- index starts -->` and `<!-- index ends -->` with the newly generated index.
- It uses `git diff` to detect if the README has changed, then if it has it runs `git commit` and `git push` to commit those changes. See my TIL Commit a file if it changed for details on that pattern.

I *really* like this pattern.

I'm a big fan of keeping content in a git repository. Every CMS I've ever worked on has eventually evolved a desire to provide revision tracking, and building that into a regular database schema is never particularly pleasant. Git solves content versioning extremely effectively.

Having a GitHub repository that can update itself to maintain things like index pages feels like a technique that could be applied to all kinds of other content-related problems.

I'm also keen on the idea of using SQLite databases as intermediary storage as part of an Actions workflow. It's a simple but powerful way for one step in an action to generate structured data that can then be consumed by subsequent steps.

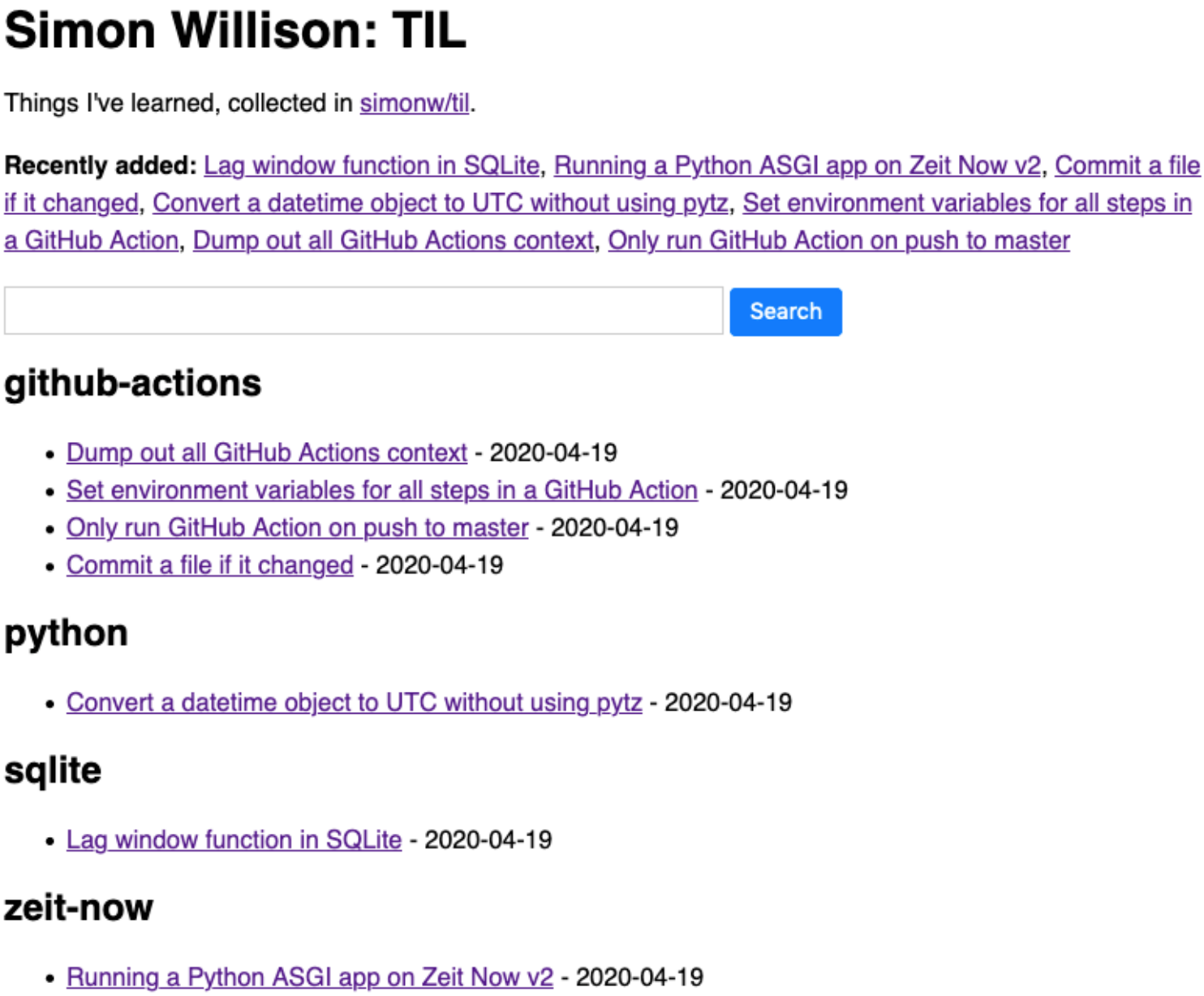
Implementing search with Datasette

Unsurprisingly, the other reason I'm using SQLite here is so I can deploy a database using Datasette. The last two steps of the workflow look like this:

```
- name: Setup Node.js
  uses: actions/setup-node@v1
  with:
    node-version: '12.x'
- name: Deploy Datasette using Zeit Now
  env:
    NOW_TOKEN: ${{ secrets.NOW_TOKEN }}
  run: |-
    datasette publish now2 til.db \
      --token $NOW_TOKEN \
      --project simon-til \
      --metadata metadata.json \
      --install py-gfm \
      --install datasette-render-markdown \
```

```
    --install datasette-template-sql \
    --template-dir templates
```

This installs Node.js, then uses Zeit Now (via datasette-publish-now) to publish the generated `til.db` SQLite database file to a Datasette instance accessible at til.simonwillison.net.



Simon Willison: TIL

Things I've learned, collected in simonw/til.

Recently added: Lag window function in SQLite, Running a Python ASGI app on Zeit Now v2, Commit a file if it changed, Convert a datetime object to UTC without using pytz, Set environment variables for all steps in a GitHub Action, Dump out all GitHub Actions context, Only run GitHub Action on push to master

Search

github-actions

- Dump out all GitHub Actions context - 2020-04-19
- Set environment variables for all steps in a GitHub Action - 2020-04-19
- Only run GitHub Action on push to master - 2020-04-19
- Commit a file if it changed - 2020-04-19

python

- Convert a datetime object to UTC without using pytz - 2020-04-19

sqlite

- Lag window function in SQLite - 2020-04-19

zeit-now

- Running a Python ASGI app on Zeit Now v2 - 2020-04-19

I'm reusing a bunch of tricks from my Niche Museums website here. The site is a standard Datasette instance with a custom index.html template that uses datasette-template-sql to display the TILs. Here's that template section in full:

```
{% for row in sql("select distinct topic from til order by topic") %}
    <h2>{{ row.topic }}</h2>
    <ul>
        {% for til in sql("select * from til where topic = '" + row.topic + "'") %}
            <li><a href="{{ til.url }}">{{ til.title }}</a> - {{ til.created[:10] }}</li>
        {% endfor %}
    </ul>
{% endfor %}
```

The search interface is powered by a custom SQL query in metadata.json that looks like this:

```
select
    til_fts.rank,
    til.*
from til
join til_fts on til.rowid = til_fts.rowid
where
    til_fts match case
        :q
        when '' then '*'
        else escape_fts(:q)
    end
order by
    til_fts.rank limit 20
```

A custom query-til-search.html template then renders the search results.

A powerful combination

I'm pretty happy with what I have here—it's definitely good enough to solve my TIL publishing needs. I'll probably add an Atom feed using datasette-atom at some point.

I hope this helps illustrate how powerful the combination of GitHub Actions, Datasette and Zeit Now or Cloud Run can be. I'm running an increasing number of projects on that combination, and the price, performance and ease of implementation continue to impress.

Posted 20th April 2020 at 1:38 am · Follow @simonw on Twitter

githubactions 38 | datasette 314 | projects 272 | markdown 12 | github 104 | til 5

Next: Weeknotes: Datasette 0.40, various projects, Dogsheep photos

Previous: Weeknotes: Hacking on 23 different projects

> *Using a self-rewriting README powered by GitHub Actions to track TILs https://t.co/BKjgNgHsHs* — Simon Willison (@simonw) April 20, 2020