Flattening a Site: From Database to Static Files

Friday, November 11th, 2016

I just finished converting a site[1] from running on a database-driven CMS (Textpattern in this case) to a bunch of static HTML files. No, I don't mean I switched to a static site generator like Jekyll or Octopress, I mean it's just plain HTML files and nothing else. I call this "flattening" a site.[2]

In this form, a web site can run for decades with almost no maintenance or cost. It will be very tedious if you ever want to change it, but that is fine because the whole point is long-term preservation. It's a considerate, responsible thing to do with a website when you're pretty much done updating it forever. Keeping the site online prevents link rot, and you never know what use someone will make of it.

How to Flatpack

Before getting rid of your site's CMS and its database, make use of it to simplify the site as much as possible. It's going to be incredibly tedious to fix or change anything later on so now's the time to do it. In particular you want to edit any templates that affect the content of multiple pages:

- Strip out all external dependencies, such as TypeKit and any script-based analytics. In place of Typekit I used a self-hosted font for the headings and just switched to Georgia for the body text.
- Removed unneeded sections and internal links from the page templates. For example, on my site I eliminated any reference to the privacy policy and the "different ways to subscribe" guide. I also made the "episodes" page one giant list of all the episodes instead of being broken up into 20 pages.
- Finally, wherever appropriate, edited text in the page templates to make the site's "archival status" clear. Remove anything that could give the impression that this place is still a going concern.

1. https://howellcreekradio.com
2. I wanted a way to refer to this process that would distinguish it from "archiving", which to me also connotes taking the site offline. I passed on "embalming" and "mummifying" for similar reasons.

Next, on your web server, make a temp directory (outside the site's own directory) and download static copies of all the site's pages into it with the `wget` command:

```
wget --recursive --domains howellcreekradio.com --html-extension howellcreekradio.com/
```

This will download every page on the site and every file linked to on those pages. In my case it included images and MP3 files which I didn't need. I deleted those until I had only the `.html` files left.

Digression: Mass-editing links and filenames from the command line

This bit is pretty specific to my own situation but perhaps some will find it instructive. At this point I was almost done, but there was a bit of updating to do that couldn't be done from within my CMS. My home page on this site had "Older" and "Newer" links at the bottom in order to browse through the episodes, and I wanted to keep it this way. These older/newer links were generated by the CMS with POST-style URLS: `http://site.com/?pg=2` and so on. When `wget` downloads these links (and when the `–html-extension` option is invoked), it saves them as files of the form `index.html?pg=2.html`. These all needed to be renamed, and the pagination links that refer to them needed to be updated.

I happen to use ZSH, which comes with an alternative to the standard `mv` command called `zmv` that recognizes patterns:

```
zmv 'index.html\?pg=([0-9]).html' 'page$1.html'
zmv 'index.html\?pg=([0-9][0-9]).html' 'page$1.html'
```

So now these files were all named `page01.html` through `page20.html` but they still *contained* links in the old `?pg=` format. I was able to update these in one fell swoop with a one-liner:

```
grep -rl \?pg= . | xargs sed -i -E 's/\?pg=([0-9]+)/page\1.html/g'
```

To dissect this a bit:

- `grep -rl \?pg= .` lists all files containing the links I want to change. I pass this list to the next command with the pipe | character.
- The `xargs` command takes the list produced by `grep` and feeds them one by one to the `sed` command.

- The `sed` command has the `-i` option to edit the files in-place, and the `-E` option to enable regular expressions. For every file in its list, it uses `s/\?pg=([0-9]+)/page\1.html/g` as a regex-style search-and-replace pattern. You can learn more about the details of this search pattern[3] if you are new to regular expressions.

OK, digression over.

Back up the CMS and Database

Before actually switching, it's a good idea to freeze-dry a copy of the old site, so to speak, in case you ever needed it again.

Export the database to a plain-text backup:

```
mysqldump -u username -pPASSWORD db_name > dbbackup.sql
```

Then save a gzip of that `.sql` file and the whole site directory before proceeding.

Shutting down the CMS and swapping in the static files

Final steps:

1. Move the HTML files you downloaded and modified above into the site's public folder.
2. Add redirects or rewrite rules for every page on your site. For example, if your server uses Apache, you would edit the site's `.htaccess` file so that URLs on your site like `site.com/about/` would be internally rewritten[4] as `site.com/about.html`. This is going to be different depending on what CMS was being used, but **essentially you want to be sure that any URL that anyone might have used as a link to your site continues to work**.
3. Delete all CMS-related files from your site's public folder (you saved that backup, right?) In my case I deleted `index.php`, `css.php`, and the whole `textpattern/` directory.

3. https://regex101.com/r/rXuSJB/1
4. http://httpd.apache.org/docs/2.0/misc/rewriteguide.html

Once you're done

Watch your site's logs for 404 errors for a couple of weeks to make sure you didn't miss anything.

What to do now? You could leave your site running where it is. Or, long term, consider having it served from a place like NearlyFreeSpeech[5] for pennies a month.

5. https://www.nearlyfreespeech.net