Administrative Tasks

This documentation is for anyone managing the repo to remember how to do occasional maintenance tasks.

Update the `rustc` version

- Delete your `target` directory, you're about to recompile everything anyway
- Change the version number in `.github/workflows/main.yml`
- Change the version number in `rust-toolchain`, which should change the version you're using locally with `rustup`
- Change the version number in `src/title-page.md`
- Run `./tools/update-rustc.sh` (see its commented code for details on what it does)
- Inspect the changes (by looking at the files changed according to git) and their effects (by looking at the files in `tmp/book-before` and `tmp/book-after`) and commit them if they look good
- Grep for `manual-regeneration` and follow the instructions in those places to update output that cannot be generated by a script

Update the `edition` in all listings

To update the `edition = "[year]"` metadata in all the listings' `Cargo.toml`s, run the `./tools/update-editions.sh` script. Check the diff to make sure it looks reasonable, and in particular check whether the updates necessitate any changes to the text. Then commit the changes.

Update the `edition` in mdBook config

Open `book.toml` and `nostarch/book.toml` and set the `edition` value in the `[rust]` table to the new edition.

Release a new version of the listings

We now make `.tar` files of complete projects containing every listing available [as GitHub Releases](https://github.com/rust-lang/book/releases). To create a new release artifact, for example if there have been code changes due to edits or due to updating Rust and `rustfmt`, do the following:

- Create a git tag for the release and push it to GitHub, or create a new tag by going to the GitHub UI, [drafting a new release](https://github.com/rust-lang/book/releases/new), and entering a new tag instead of selecting an existing tag
- Run `cargo run --bin release_listings`, which will generate `tmp/listings.tar.gz`
- Upload `tmp/listings.tar.gz` in the GitHub UI for the draft release
- Publish the release

Add a new listing

To facilitate the scripts that run `rustfmt` on all the listings, update the output when the compiler is updated, and produce release artifacts containing full projects for the listings, any listing beyond the most trivial should be extracted into a file. To do that:

- Find where the new listing should go in the `listings` directory.
  - There is one subdirectory for each chapter
  - Numbered listings should use `listing-[chapter num]-[listing num]` for their directory names.
  - Listings without a number should start with `no-listing-` followed by a number that indicates its position in the chapter relative to the other listings without numbers in the chapter, then a short description that someone could read to find the code they're looking for.
  - Listings used only for displaying the output of the code (for example, when we say "if we had written x instead of y, we would get this compiler error:" but we don't actually show code x) should be named with `output-only-` followed by a number that indicates its position in the chapter relative to the other listings used only for output, then a short description that authors or contributors could read to find the code they're looking for.
  - **Remember to adjust surrounding listing numbers as appropriate!**
- Create a full Cargo project in that directory, either by using `cargo new` or copying another listing as a starting point.
- Add the code and any surrounding code needed to create a full working example.
- If you only want to show part of the code in the file, use anchor comments (`// ANCHOR: some_tag` and `// ANCHOR_END: some_tag`) to mark the parts of the file you want to show.
- For Rust code, use the `{{#rustdoc_include [filename:some_tag]}}` directive within the code blocks in the text. The `rustdoc_include` directive gives the code that doesn't get displayed to `rustdoc` for `mdbook test` purposes.
- For anything else, use the `{{#include [filename:some_tag]}}` directive.

- If you want to display the output of a command in the text as well, create an
  `output.txt` file in the listing's directory as follows:
  - Run the command, like `cargo run` or `cargo test`, and copy all of the
    output.
  - Create a new `output.txt` file with the first line `$ [the command you
  ran]`.
  - Paste the output you just copied.
  - Run `./tools/update-rustc.sh`, which should perform some normalization on
    the compiler output.
  - Include the output in the text with the `{{#include [filename]}}` directive.
  - Add and commit output.txt.
- If you want to display output but for some reason it can't be generated by a
  script (say, because of user input or external events like making a web
  request), keep the output inline but make a comment that contains
  `manual-regeneration` and instructions for manually updating the inline
  output.
- If you don't want this example to even be attempted to be formatted by
  `rustfmt` (for example because the example doesn't parse on purpose), add a
  `rustfmt-ignore` file in the listing's directory and the reason it's not
  being formatted as the contents of that file (in case it's a rustfmt bug that
  might get fixed someday).

See the effect of some change on the rendered book

To check, say, updating `mdbook` or changing the way files get included:

- Generate a built book before the change you want to test by running `mdbook

  build -d tmp/book-before`
- Apply the changes you want to test and run `mdbook build -d tmp/book-after`
- Run `./tools/megadiff.sh`
- Files remaining in `tmp/book-before` and `tmp/book-after` have differences
  you can manually inspect with your favorite diff viewing mechanism

Produce new markdown files for No Starch

- Run `./tools/nostarch.sh`
- Spot check the files that script created in the `nostarch` directory
- Check them into git if you're starting a round of edits

Produce markdown from docx for diffing

- Save the docx file to `tmp/chapterXX.docx`.
- In Word, go to the review tab, choose "Accept all changes and stop tracking"
- Save the docx again and close Word
- Run `./tools/doc-to-md.sh`
- This should write `nostarch/chapterXX.md`. Adjust the XSL in
  `tools/doc-to-md.xsl` and run `./tools/doc-to-md.sh` again if needed.

Generate Graphviz dot

We're using [Graphviz](http://graphviz.org/) for some of the diagrams in the book. The source for those files live in the `dot` directory. To turn a `dot` file, for example, `dot/trpl04-01.dot` into an `svg`, run:

```bash
$ dot dot/trpl04-01.dot -Tsvg > src/img/trpl04-01.svg
```

In the generated SVG, remove the width and the height attributes from the `svg` element and set the `viewBox` attribute to `0.00 0.00 1000.00 1000.00` or other values that don't cut off the image.

Publish a preview to GitHub Pages

We sometimes publish to GitHub Pages for in-progress previews. The recommended flow for publishing is:

- Install the `ghp-import` tool by running `pip install ghp-import` (or `pipx install ghp-import`, using [pipx][pipx]).
- In the root, run `tools/generate-preview.sh`

[pipx]: https://pipx.pypa.io/stable/#install-pipx