# Guide to the research lifecycle in DP-Next

Luke W. Johnston

# Contents


Welcome! . . . 5
Who you are . . . 5
Contributing . . . 5
Changes . . . 5
How the website is made . . . 5
Inspirations . . . 6

I Overview

1 Overview . . . 9

2 Roles and responsibilities . . . 11
2.1 Roadmap management . . . 11
2.2 Issue management . . . 11

II Lifecycle

3 Getting started . . . 15

4 Start a new project . . . 17

5 Denmark Statistics . . . 21
5.1 Configure Git . . . 21
5.2 The folder structure . . . 21
5.3 Creating the “parent” folder . . . 22
5.4 Creating a Git “remote” as a folder . . . 22
5.5 Creating the first “local” repository . . . 22
5.6 Creating more “local” repositories . . . 23
5.7 Using the “remote” to collaborate . . . 23

6 Authorship . . . 25

Appendices . . . 27

Changelog . . . 29
0.2.0 - 2026-08-30 . . . 29
✨ Features . . . 29
💄 Styling . . . 29
0.1.1 - 2026-08-30 . . . 29

🐛 Fixes . . . 29
👷 CI/CD . . . 29
**0.1.0 - 2026-08-30** . . . **29**
✨ Features . . . 29
🐛 Fixes . . . 29
♻️ Refactor . . . 29
📝 Documentation . . . 29
💄 Styling . . . 30
🧑‍💻 Miscellaneous . . . 30
❤️ New contributors . . . 30

## Contributing . . . 31

**:bug: Issues and bugs** . . . **31**
**:pencil2: Adding or modifying content** . . . **31**
**:file_folder: Explanation of files and folders** . . . **31**

## Contributor Code of Conduct . . . 33

# Welcome!

> **Note**
>
> This guide mostly follows the diátaxis "how-to guide" style, though not strictly. It is a living and constantly evolving guide that is regularly updated as we learn and refine how we work and develop data packages. We intend to continually update and release every update to the guide to Zenodo and as GitHub releases. We don't expect this guide to ever be considered "done".

There are surprisingly few, practical resources and how-to guides for learning about and doing effective collaboration within research, especially in the context of open science and reproducibility. The few resources that do exist are more theoretical than the practice "how do I actually do it?" aspects.

In particular, few research groups and institutions develop and curate their own guide on how they do things. This makes it very difficult to learn as a new (or established) researcher how others do things, so that we can draw inspiration from or build off of how others work and do research. Often these skills are gained through verbal knowledge sharing or from watching how peers or other group members work. This makes it difficult to apply any type of scientific reasoning, critical thinking, and analysis to these collaborative, open scientific, and reproducible practices.

This guide is an attempt at documenting and sharing how we aspire to and aim to work within the DP-Next project.

## Who you are

We've written these documents with a few types of people in mind:

- **DP-Next collaborators/group members**: These people are our primary audience for this guide. If you work with us in DP-Next, these documents are designed and developed with you in mind.
- **Researchers not affiliated with DP-Next**: If you work in another organization or group, we write these documents to share our practices with you. You can use this either as a guide or as a reference to help you improve your own work or to draw inspiration from for how you structure your research and collaboration.
- **Group or organisation leaders**: If you are a leader of a group or institution, we write this guide to show how a group of researchers *could* work together in more effective ways. You can use this guide to get inspiration for how you can design your own group or institution to work better together at producing open and reproducible research.

## Contributing

Check out our contributing document for information on how to contribute to this guide.

## Changes

Check out our changelog for information on what has changed in each version of the guide. Previous versions can be found in the releases page of the GitHub repository.

## How the website is made

The website and PDF (download link on the top of the website's sidebar) are created using Quarto to write the material and create the book format. GitHub hosts the Git repository of the material, while GitHub Actions builds and releases the book with every change. Netlify hosts the website and manages the domain. The source material is in the `guide` GitHub repository.

## Inspirations

- The Turing Way

# I Overview


1 Overview . . . . . 9

2 Roles and responsibilities . . . . . 11
2.1 Roadmap management . . . . . 11
2.2 Issue management . . . . . 11

# 1. Overview

# 2. Roles and responsibilities

In a larger project there are many tasks and needs to be done, many more than any individual person can handle at any one time. To manage this, we describe specific responsibilities that need to be done within DP-Next and assign them to people so that people's roles are clear and they know what they are responsible for.

## 2.1 Roadmap management

A roadmap is a set of products, milestones, and timelines necessary to complete a particular project. In our case, a project would be an individual work package.

An effectively developed and regularly maintained roadmap is critical for successful and timely completion of a project. But they aren't just for planning purposes, they also are extremely effective at communicating the project's status, next steps, progress, and direction to everyone involved in the project. It keeps everyone aligned and makes it clearer what needs to be done.

Because roadmaps are so important and require regular maintenance and management, someone needs to be responsible for them. The roadmap manager's tasks are:

- Identify all concrete products for a project (one product is one repository) by talking to the collaborators and list them in the roadmap (see how below). This is a critical step, but also can be extremely difficult to do, as knowing clearly what a product is and what products are needed for the project can be hard for collaborators to themselves communicate and agree on. This is part of the responsibility of the roadmap manager to dig in and ask detailed and probing questions. This will be an iterative process, so don't expect to get clear answers right away.
- Identify all milestones for each product, again by talking to the collaborators by probing and digging in to get as clear an idea and picture as possible on what needs to be done for each product. Then list them in the roadmap. Like the products, this is something that will need to be done regularly and will evolve over time. So expect to regularly talk with collaborators about milestones and tasks.

We use GitHub Projects to create and manage roadmaps (see some of our roadmaps in our list of project boards). These are some things that we do in the GitHub Projects to manage our roadmaps:

- Use the roadmap view in GitHub Projects as the main view for the roadmap project board. Group this view by "product" (see below) to see the milestones (see below) grouped by product. This main view should be filtered by `-status:Done`, so that only active milestones are shown.
- Create draft issues and use them to represent milestones for a given product. Start and end dates should be put on each milestone.
- Create a custom, single-select field to represent the project's products and within that field, add options for each of the products in the project. One product should represent one repository. Each milestone ("draft issue") should be assigned to a product using this single-select "products" field, so that milestones are grouped by product in the roadmap view.
- When a milestone is complete, change the status to "Done" so it gets filtered from the main roadmap view.

## 2.2 Issue management

GitHub Issues are the primary way we track work in DP-Next. We use issues to describe specific tasks that need to be done, to have a place to discuss and brainstorm about particular topics, and come to an agreement or decision for those topics. As a product progresses, more issues are made, often faster than they can be completed. And issues can quickly accumulate, go out-of-date, or be forgotten. Combined with the fact that the priority of any given issue needs to reflect the current state of the roadmap, someone needs to be responsible for curating and managing the issues for a given product. The issue manager's tasks are:

- Understand the product enough to create issues that contribute to its completion.

- Track and ensure issues are completed and closed, by reminding assignees of their issues and checking on the status of issues regularly (ideally during update meetings)
- Close any issues that are no longer relevant or are out-of-date.
- Consult the roadmap to identify what issues need to be prioritized in any given period of time.
- Ensure that the distribution of the issues priorities are balanced. If all issues are high priority, no issue is a high priority. There needs to be a balance between high, medium, and low priority as it reflects an understanding the actual state and needs of the product and overall project.

We use GitHub Issues as well as GitHub Projects to manage our issues. These are some things that we do in GitHub to manage our issues:

- Use a Board view in GitHub Projects, grouped by "status" to show "todo", "in progress", and "done" columns. This is the main view for managing issues. Create "swimlanes" on the board by assignee so each person has their own "lane" to track and manage their own issues.
- Create a custom, single-select field to represent the issue's priority and within that field, add options for "high", "medium", and "low" priority.
- Assign priorities to issues using the "priority" field. An easy way to do this is to create a new view (just for the issue manager) that has all the issues grouped by priority. With the grouping, it is easy to drag issues between the different priority levels to assign them. This also makes it easy to see the distribution of priorities across all issues and ensure that they are balanced.
- Keep the main view (grouping by status and swimlane by assignee) filtered by `has:priority`, so that only issues with priorities are shown. This helps keep the focus on and communicates to everyone what actually needs to get done.

# II Lifecycle


3 Getting started . . . . . 15

4 Start a new project . . . . . 17

5 Denmark Statistics . . . . . 21
5.1 Configure Git . . . . . 21
5.2 The folder structure . . . . . 21
5.3 Creating the “parent” folder . . . . . 22
5.4 Creating a Git “remote” as a folder . . . . . 22
5.5 Creating the first “local” repository . . . . . 22
5.6 Creating more “local” repositories . . . . . 23
5.7 Using the “remote” to collaborate . . . . . 23

6 Authorship . . . . . 25

# 3. Getting started

# 4. Start a new project

If you are part of DP-Next (or not), you'll be working on research projects, likely using data collected during DP-Next or from existing data (e.g. ADDITION-PRO). This chapter describes what to do when starting a new project. Below in Figure 4.1 is a visual summary of the steps to take, along with descriptions afterwards that outline the steps in more detail.

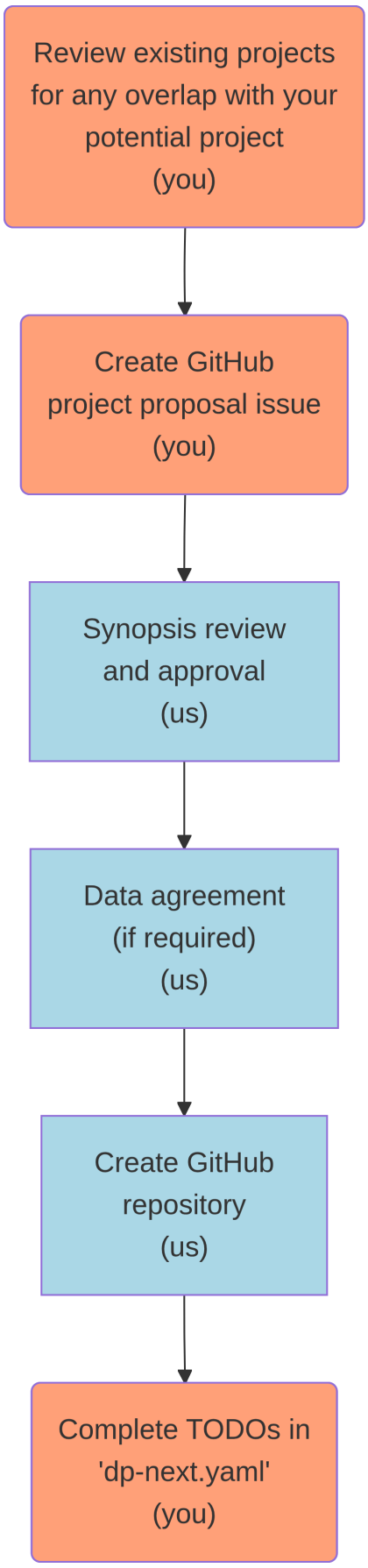



Figure 4.1: Steps to take for starting a project within DP-Next. The rounded boxes in orange are tasks you are responsible for doing. The rectangular ones in light blue are for the managers ("us") to do.

The requirements for preparing a DP-Next publication apply to both internal and external researchers and should follow these steps:

1. Before proposing a project, make sure that you aren't doing something that's already done or being worked on. If there is a current ongoing project that is similar to what you wanted to work on, you can request to collaborate on that project with those researchers.
2. If there is no other project that is similar to your project idea, you can propose the project. Create a new issue in the WP1-Management repo (private for now) and select the "Project proposal" issue template. Fill in the fields there, which we'll use to assess how relevant the project is within DP-Next.
3. Once you've created the issue, at least one member of the Steering Committee needs to review and approve it (usually either the principle investigator or the general coordinator). It is possible that someone may approve it, but later another committee member may request changes or give comments. If that happens, it is fairly easy to update the project. Once approved, someone with access permissions (e.g. in WP1) will create a GitHub repository for the project by using the DP-Next template project and following our naming convention.
4. After the repository is created, you need to go through the repository and complete any TODO items, such as those found in the `TODO.md` file. Importantly, the TODO items in the `dp-next.yaml` file need to be filled. We use the information in this file to create an overview of the different projects going on in DP-Next.

> **Note**
>
> We expect that many new projects based on DP-Next data will be done by DP-Next researchers. However, we also welcome collaborations with external researchers (i.e. those who are not part of DP-Next), though these projects must include at least one DP-Next member as coauthor so that projects follow our standards. Projects can be stand-alone or part of a wider initiative, such as a PhD or postdoc protocol.

> **Warning**
>
> For investigators not covered by the DP-Next agreements (by institutional affiliation to one of the Steno Diabetes Centres), a separate data agreement must be signed if they require data access. This agreement must be in place before any data access can be given.

# 5. Denmark Statistics

Working on the Denmark Statistics (DST) servers brings with it unique challenges when it comes to collaborating together. This page describes how we've set it up to effectively collaborate on the DST servers.

> **i** Note
>
> We show the steps to take using the terminal. Why? Because a lot of power and effectiveness comes from using the terminal and some steps can only be done in the terminal. It is also just easier to show a series of commands to run in the terminal rather describe how to do it with R, which requires more steps to take (e.g. opening up RStudio and finding the right working directory).

## 5.1 Configure Git

The very first step is to configure your Git. You thankfully only need to do once (or after DST has reset your profile). If you haven't already configured your Git, open a terminal (called Git Bash in DST) and type out these commands:

```
git config --global user.name "YOUR NAME"
git config --global user.email "name@email.com"
git config --global init.defaultBranch main
```

Replacing the `YOUR NAME` and `name@email.com` with your actual details.

## 5.2 The folder structure

Before setting up this folder structure, the first thing to decide is what the "product"/output will be and what its name (or abbreviation) will be. The reason for this, as per our design principles, is that we want to structure work around products and around teams (or groups), rather than individual people. So a folder structure should reflect how to work on the product as a group.

We'll use an example of a product called `wp2-analysis` that will be done within DST. There are two people working on this product (for now) called `person-name-1` and `person-name-2`. The folder structure will look like this (don't create it yet!):

```
workdata/<project-id>/
└── wp2-analysis/
    ├── remote/ # Contains only Git specific folders
    ├── person-name-1/
    │   ├── .git/
    │   ├── data/
    │   ├── docs/
    │   ├── R/
    │   ├── wp2-analysis.Rproj
    │   ├── DESCRIPTION
    │   ├── _targets.R
    │   └── README.md
    └── person-name-2/
        ├── .git/
        ├── data/
        ├── docs/
        ├── R/
        ├── wp2-analysis.Rproj
```

```
        ├── DESCRIPTION
        ├── _targets.R
        └── README.md
```

The `remote/` folder is a (bare) Git repository, meaning that the only files and folders found within it are Git specific folders and files, such as `branches/`, `hooks/`, `objects/`, `refs/`, `config`, `description`, and `HEAD`. The other folders (for each collaborator) are the "local" repositories that have the standard structure for data analysis projects. In this case, the "remote" folder is a bit like a GitHub repository, while each collaborator's "local" repository is like their own local repository of the GitHub (remote) repository.

> ! Important
>
> We'll describe the steps later on, but an important note is that each individual person will need to create their own folder. That's because servers like DST assigns the "owner" of the folder to the user who created it, so you may encounter issues if someone else made the folder for you.

## 5.3 Creating the "parent" folder

You need to first open a terminal in the right location and create the correct folder. To do that, open up the project's `workdata/` folder using the File Explorer in the DST Windows desktop. The project folder should be at a location like `E:/workdata/7000000/` (the project ID is usually a seven-digit number starting with 7). When in this folder, right-click somewhere in the folder and select the menu option "Open in Git Bash" (or something similar). A terminal program should open up at this location. In the new terminal app, create the new research project folder (`wp2-analysis` from the example above):

```
mkdir wp2-analysis
```

The `mkdir` command means "make directory" (or folder). Next, you want to move your "working directory" (the folder location your terminal is in) into this newly created folder using the `cd` command (change directory):

```
cd wp2-analysis
```

You are now ready for the next steps.

## 5.4 Creating a Git "remote" as a folder

To create the "remote" folder, first create the `wp2-analysis/` folder. Then, open a terminal into the `wp2-analysis/` folder and run the following command:

```
git init --bare remote
```

This will then create the `remote/` folder with the necessary Git files and folders that will allow it to be treated as a Git remote repository (like with GitHub). This is the folder each collaborator will push and pull to (more on that later below).

## 5.5 Creating the first "local" repository

Next, one of the collaborators (e.g. `person-name-1`) needs to create their local repository, ideally using a standard structure as set up by the prodigenr package. To do this run the following R function while in the terminal:

```
Rscript -e 'prodigenr::setup_project("wp2-analysis")'
```

This will create a folder with the `.Rproj` file name as `wp2-analysis`. But since we want each subfolder to be the name of the collaborator, you'll need to rename the folder to `person-name-1` (name or other identifier of the person doing these tasks, without spaces). You can use the `mv` command in the Git Bash terminal, which renames a file or folder to a new name using the pattern `mv OLD NEW`.

```
mv wp2-analysis person-name-1
```

Next, you’ll need to connect this local repository to the `remote/` folder. Before you do that, you’ll need to “move into” that new folder by using the “change directory” command `cd`.

```
cd person-name-1
```

Next, you can add all the files in that directory into the Git history:

```
git add .
git commit -m "start of project"
```

Now that this folder has files saved to the Git history, you can connect the repository to the `remote/` folder.

```
git remote add origin ../remote
```

You’ll use the relative path `../remote` to mean “one folder up” (`../`) as you will always be keeping both the remote and “local” (collaborator) repositories within the same parent folder (e.g. `wp2-analysis/`).

Next you’ll need to push the Git history to the remote, while also telling the project’s Git to use that remote whenever you use push and pull with the `-u` option:

```
git push -u origin main
```

You’re now ready to collaborate! :tada:

## 5.6 Creating more “local” repositories

Now that the “remote” folder has been officially set up, it’s easy to add another “local” repository for another collaborator (e.g. `person-name-2`). Replace `person-name-2` with the name (or other identifier) of the second collaborator (without spaces). This step should be done by the other collaborator (e.g. the person who’s name is `person-name-2`). You’ll need to open the terminal in the `wp2-analysis` folder. You can check if you are in the right folder by using the `pwd` command (print working directory).

```
pwd
```

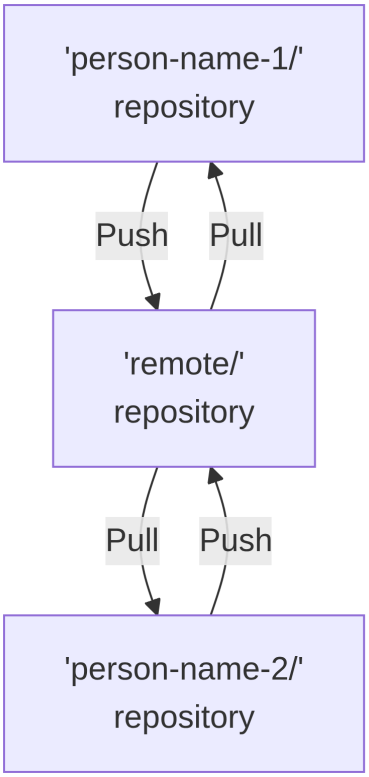
And it should output: `E:/workdata/PROJECT-ID/wp2-analysis`. If you are *not* at that location, make sure to open the File Explorer, find that folder, and right-click somewhere to select “Open in Git Bash” in the menu. When you’re in the right folder, run this command:

```
git clone ./remote person-name-2
```

This will create a new folder called `person-name-2/` with the same structure as the first collaborator’s local repository. It will also be connected to the same `remote/` folder for pushing and pulling changes. This collaborator can now open the project in RStudio, Positron, or any other editor they like.

## 5.7 Using the “remote” to collaborate

Now that you have the remote and multiple local folders for each collaborator, you can start collaborating together! Whenever either collaborator makes changes to their local repository, regularly “push” and “pull” to the remote (via RStudio’s interface or via the terminal) so that the remote is always up to date. This can be visually represented as follows:




This way, both collaborators can work on their own local repositories, but they can easily share their changes with each other by pushing and pulling to the remote repository. It also means that any collaborator can contribute to the product repository without impacting the work of other collaborators. This allows for effective collaboration while working on the DST servers, even without access to external platforms like GitHub.

There will of course be some issues that will come up, especially merge conflicts. If each collaborator is pushing and pulling to the `main` (could also be called `master`) branch, then you will likely encounter merge conflicts. This isn't necessarily a bad thing, as you can always resolve them. And it's also fairly easy to avoid if you communicate with each other and coordinate your work effectively, e.g. by saying "hey I'm working on this file".

A more powerful approach is to use branches for each change. This is most commonly used when working with GitHub-type repositories. In this case, it might not be necessary, nor is it as easy to do within the DST environment. So for now, the best approach is to just coordinate and communicate with each other while pushing and pulling to the `main` branch.

# 6. Authorship

Authorship publications should reflect a substantial, direct contribution to the work presented in each paper. We aim to use the Vancouver Guidelines as a base and expand on them to reflect nontraditional research contributions. Nontraditional research contributions are activities that contribute in a direct way to the creation of the publication, such as improving data quality, developing software, or improving analysis code.

When you create the project proposal (as in Chapter 4.), you should have an approximate idea of who the authors are and their order in the list. As the project continues, though, this could naturally change. Anyone who has contributed meaningfully or substantially to research project should be given the chance to become coauthor.

All potential coauthors should be involved early in the research project, ideally at the project proposal stage. In addition to the named authors, all papers need to include the group authorship of 'on behalf of the DP-Next investigators', linking to the current list of DP-Next investigators

Keep in mind that author lists and orders aren't fixed at the start and can be amended after project approval by consensus among the authors or by the Steering Committee if the factual contributions differ from those foreseen initially. Ultimately, the Steering Committee has the final authority to decide on and determine the author list of any given paper. If disagreements about authorship cannot be resolved by consensus within the author group, the Steering Committee will make the final decision.

Regardless of named authors, you should make sure to include a detailed contributors list along with their described contributions as per the CRediT roles. At the least, the contributors and CRediT roles should be kept updated in the `dp-next.yaml` file and ideally be mentioned somewhere in the paper or in the supplemental material. The Steering Committee will ensure that such a document is publicly accessible and up to date.

Make sure to also include an acknowledgement to external parties such as funders, institutions, participants, and service providers in papers and other material you present or disseminate elsewhere. The Steering Committee will maintain an updated acknowledgement statement for the DP-Next project in the template-project.

# Appendices

# Changelog

Since we follow Conventional Commits, we're able to automatically create formal "releases" of the website based on our commit messages. Releases in the context of websites are simply snapshots in time of the website content. We use Cocogitto to be able to automatically create these releases, which uses SemVar as the version numbering scheme, and git-cliff to generate the changelog based on the commit messages.

Because releases are created based on commit messages, a new release is created quite often—sometimes several times in a day. This also means that any individual release will not have many changes within it. Below is a list of the releases we've made so far, along with what was changed within each release.

Commits from bots, like `dependabot` or `pre-commit-ci`, are not included in the changelog.

## 0.2.0 - 2026-08-30

### ✨ Features

- Move authorships section over from `wp1-ros` #4 by `@lwjohnst86` (ce029c1)

### 💄 Styling

- Don't number appendices #1 by `@lwjohnst86` (e876d30)

## 0.1.1 - 2026-08-30

### 🐛 Fixes

- Remove `.` before `filename` in code chunks by `@lwjohnst86` (116392c)

### 👷 CI/CD

- Fix malformed workflow by `@lwjohnst86` (278fce3)

## 0.1.0 - 2026-08-30

### ✨ Features

- Add initial Quarto config file and index files by `@lwjohnst86` (1b3dcea)
- Move over section on starting a project by `@lwjohnst86` (e7c19e6)
- Move "roles" chapter into this guide by `@lwjohnst86` (29b4021)
- Move and update the DST setup guide by `@lwjohnst86` (d5e8640)

### 🐛 Fixes

- Add link to Turing Way website by `@lwjohnst86` (a5f2e85)

### ♻️ Refactor

- Rename to `overview` from `general` by `@lwjohnst86` (1a71f31)

### 📝 Documentation

- Minor edits to community health files by `@lwjohnst86` (41f4afd)

## 💄 Styling

- Add DP-Next Quarto theme by `@lwjohnst86` (fb45ae0)

## 👨‍💻 Miscellaneous

- Start of guide repo by `@lwjohnst86` (8feebcc)
- Switch to use `dp-next-theme` in justfile by `@lwjohnst86` (06fce32)
- Update and run pre-commit hooks by `@lwjohnst86` (f937cf2)
- Exclude `_book/` and `_extensions/` from TODO listing by `@lwjohnst86` (ae8107a)
- Ignore Quarto `*_files/` by `@lwjohnst86` (102e2ed)

## ❤️ New contributors

- `@github-actions[bot]` started making automated contributions
- `@lwjohnst86` made their first contribution

# Contributing

## :bug: Issues and bugs

The easiest way to contribute is to report issues or bugs that you might find while reading through the website. You can do this by creating a new issue on our GitHub repository.

## :pencil2: Adding or modifying content

If you would like to contribute content, please check out our guidebook for more specific details on how we work and develop. It is a regularly evolving document, so is at various states of completion.

To contribute to `guide`, you first need to install uv and justfile. We use uv and justfile to manage our project, such as to run checks and test the template. Both the uv and justfile websites have a more detailed guide on using uv, but below are some simple instructions to get you started.

It's easiest to first install uv and then install justfile with uv. Once you've installed uv, install justfile by running:

```
uv tool install rust-just
```

We keep all our development workflows in the `justfile`, so you can explore it to see what commands are available. To see a list of commands available, run:

```
just
```

As you contribute, make sure your changes will pass our tests by opening a terminal so that the working directory is the root of this project's repository and running:

```
just run-all
```

When committing changes, please try to follow Conventional Commits as Git messages. Using this convention allows us to be able to automatically create a release based on the commit message by using Cocogitto. If you don't use Conventional Commits when making a commit, we will revise the pull request title to follow that format. That's because we squash merge when merging pull requests, so all other commits in the pull request will be squashed into one commit.

## :file_folder: Explanation of files and folders

This is a description of some of the files in this repository.

- `.copier-answers.yml`: Contains the answers you gave when copying the project from the template. **You should not modify this file directly.**
- `.github/`: Contains GitHub-specific files, such as issue and pull request templates, workflows, dependabot configuration, pull request templates, and a CODEOWNERS file.
- `_quarto.yml`: Quarto configuration file for the website, including settings for the website, such as the theme, navigation, and other options.
- `_metadata.yml`: Quarto metadata file for the website, including information about the project, such as the titles and GitHub names.
- `.gitignore`: This ignore file tells Git which files to not track. Unless you know what you are doing, it's best to not touch this file.
- `.pre-commit-config.yaml`: Pre-commit configuration file for managing and running checks before each commit.
- `.config/`: Contains configuration files for various tools used in the project, such as:
  - `typos.toml`: typos spell checker configuration file.
  - `rumdl.toml` and `panache.toml`: rumdl and Panache configuration file for formatting Markdown files in the project.

  - `cog.toml`: Cocogitto configuration file for managing versions.
  - `cliff.toml`: git-cliff configuration file for creating the changelog.
- `.editorconfig`: Editor configuration file for EditorConfig to maintain consistent coding styles across different editors and IDEs.
- `.zenodo.json`: Structured citation metadata for your project when archived on Zenodo. This is used to add the metadata to Zenodo when a GitHub release has been uploaded to Zenodo.
- `justfile`: `just` configuration file for scripting project tasks.
- `CHANGELOG.md`: Changelog file for tracking changes in the project.
- `CONTRIBUTING.md`: Guidelines for contributing to the project.

# Contributor Code of Conduct

As contributors and maintainers of this project, we pledge to respect all people who contribute through reporting issues, posting suggestions, updating any material, submitting pull requests, and other activities.

We are committed to making participation in this project a harassment-free experience for everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, or religion.

Examples of unacceptable behavior by participants include the use of sexual language or imagery, derogatory comments or personal attacks, trolling, public or private harassment, insults, or other unprofessional conduct.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct. Project maintainers who do not follow the Code of Conduct may be removed from the project team.

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by opening an issue or contacting one or more of the project maintainers.

This Code of Conduct is adapted from the Contributor Covenant (https://www.contributor-covenant.org), version 1.0.0, available at https://www.contributor-covenant.org/version/1/0/0/