Git Source Repositories

Table of Contents

By setting a minimum set of standards for all git repositories we can ensure that all repositories are consistent and easy to use. This will make it easier for developers to move between projects and to understand the structure of a project and for new developers to get up to speed quickly.

These standards also support teams that support multiple teams and projects including engineering, DevOps and SRE teams.

1. Required content

WARNING

Repositories created after the standards comes into effect that do not meet the **required** standards outlined below will be disabled by marking them as archived in Github or Azure DevOps.

Planned effective date: January 2024

The project or organization owner can unarchive the reository so it can be updated.

Repositories created before the effective date will not be affected unless they are renamed.

1.1. Repository Naming

Consistent naming helps people find project repositories. At the time or writing there is no one project naming convention. Most open source projects use lowercase names separated by hyphens. Some projects use camel case.

By using a consistent naming convention we can ensure that all repositories are easy to find and that they are consistent with other repositories reducsing cognative load. Consistent naming combined with consistent content makes it easier for developers to move between projects and to understand the structure of a project and for new developers to get up to speed quickly.

A good approach to repository naming is to name repositories after the glitz asset name adding the asset area name if multiple repositories are used.

1. All lowercase, no spaces, underscores, numbers, or special characters with hyphens as separators. e.g. **asset-name** or **asset-name-area-name**
2. Mixed case .Net naming convention e.g. **MyProject.MyPackage**

Mixing hypens and periods separators should not be used. Similarly acronyms and single word repository names should be avoided.

1.2. README files

A Git repository's README file serves several important purposes:

Introduction/Overview

It provides a brief introduction to the project. This helps anyone unfamiliar with the project to understand what the project is about.

Setup Instructions

The README often contains information on how to set up or install the project. This could include instructions on any special setup required to build or run the project, including any settings for IDE setup.

Build Instructions

If the project requires building (compiling), the README provides specific instructions on how to do this.

Test Instructions

If the project has a test suite, the README provides instructions on how to run these tests.

Usage

The README can also provide a basic tutorial on how to use the project or software.

In essence, a well-written README file acts as a guide for new users and contributors, making it an essential component of any GitHub repository.

1.3. .Manifest files

Manifest files are YAML files that contain project metadata that provides a link to other information sources (Glitz/Product Showcase) and internal references to AI model information, project technical documentation and architecture decision records.

Manifest files: https://confluence.lexisnexis.dev/display/TS/Manifest+Files

1.4. Architecture Decision Record

At a minimum one ADR is required that documents why the repository is required and in partiular why the project code could not be incoportated into another project repository.

More details on how to write and manage ADRs can be found here: https://confluence.lexisnexis.dev/pages/viewpage.action?pageId=26691717

2. Recommended additional content

The required files should be included in all repositories. The following files are recommended in particular circumstances.

2.1. Backstage Catalog

A catalog-info.yaml file should be included in all repositories to provide documentation and technical details https://backstage.lexisnexis.dev/docs/default/component/lexisnexis-backstage

2.2. CODEOWNERS

For monorepos a CODEOWNERS file should be included in the root of the repository to provide guidance on who owns the repository and who should be contacted for support.

A **CODEOWNERS** file is a file that defines individuals or teams that are responsible for code in a repository. This file is used by GitHub (and other tools) to automatically assign reviewers to pull requests, and to protect certain parts of the codebase.

In a monorepo, where multiple projects live in the same repository, a **CODEOWNERS** file can be particularly useful. It allows you to specify code ownership at a granular level, down to individual directories or even specific files. This means that the right people are notified and involved when changes are proposed to parts of the code they own.

The **CODEOWNERS** file uses a syntax similar to gitignore files. Each line is a file pattern followed by one or more owners. The owners can be GitHub users or teams, specified with an @ symbol. Here's an example:

```
# This is a comment.
* @user1 ①

/project1/ @team1 ②

/file.txt @user2 ③
```

① Assigns ownership of all files in the repo to @user1

② Assigns ownership of all files in the 'project1' directory to @team1

③ Assigns ownership of the 'file.txt' in the root to @user2

In this example, any changes to files in the project1 directory will automatically request review from @team1, and any changes to file.txt will request review from @user2.

Remember, the CODEOWNERS file should be placed in the repository's root, docs, or .github directory, and it needs to be on the repository's default branch (usually main).

2.3. Contributors file

A **CONTRIBUTORS.md** file is a document that provides guidelines on how contributions can be made to the codebase. This is particularly important when more than one team is working on a project. e.g. an InnerSource project or Monorepo.

Typical contents of a CONTRIBUTORS.md file include:

Contribution Guidelines

The file can also include guidelines on how to contribute to the project. This could include steps on how to submit a pull request, coding standards to follow, how to report bugs, and so on.

For some projects it is also useful to include:

List of Contributors

This is a list of all the people who have contributed to the project. It can be in alphabetical order, in the order of contribution, or in any other order the project maintainers deem fit. Each entry usually includes the contributor's name and a brief description of their contribution.

Acknowledgements

This section is for thanking contributors for their work. It can be a simple thank you message, or it can include more personal acknowledgements.

Remember, the purpose of the CONTRIBUTORS.md file is to acknowledge the work of those who have contributed to the project. It's a way of saying thank you and giving credit where credit is due.

Contribution Guidelines: If the project is open for contributions, the README usually includes guidelines on how to contribute, which could include a link to the project's code of conduct.