NW-PaGe Github Organization Policies

Security policies, repo guides and templates for pushing code to the Github Org

Frank Aragona

2023-09-01

Table of contents

1 Introduction

This document details the policies and guidelines for the Northwest Pathogen Genomics Center of Excellence (NW-PaGe) Github Organization.

For more information, tutorials and code examples, please see the policies website here https://nw-page.github.io/standards/.

1.1 Security

Personal or network credentials, tokens, server names and more are prohibited from being published to our public repos. Protect security and credentials with good coding practices. Using local environment files along with `.gitignore` can prevent credentials from being accidentally pushed into your repo. Other guardrails like **pre-commit hooks** can be used to further prevent accidental credential leaks. **See more details on our security policies here**

Listing 1 `local-credentials.yml`

```yaml
# local yaml file that will NOT be pushed to the repo
# add this file to the .gitignore to prevent leaks
my_credentials:
  username: super_secret_user_name
  password: 12345terriblepassword
```

Listing 2 `script-in-repo.R`

```r
# this script is in the repo, but credentials are hidden
library(yaml)

# read in the local credentials yaml file
creds <- yaml::read_yaml("path/to/local_credentials.yml")

# pull in the credentials
username <- creds$my_credentials$username
password <- creds$my_credentials$password
```

1.2 Licensing

Each repo in the organization should have a license. Licenses can help prevent your work from being stolen and/or used inappropriately. Licensing details here

1.3 Policies

This organization requires each repo to have certain documents, such as a CODE_OF_CONDUCT. The organization uses a special repo called **.github** to ensure all repos adhere to the policies and have the right documents.

1.4 Guides

If you're looking to build a repository, check out the Github user guide tabs for best practices on reproducibility, documentation in the repo, and more.

2 Security

Objectives

- Prevent sensitive information leaks to Github
- Set up guardrails, `.gitignore`, hooks
- Scrub private repos before they go public

If sensitive information is leaked and commited to the remote repo, then they will stay in the git history (and will require a lot of effort to remove them from the history). The following cannot be included in any repo **or any local commit!**:

Type	Examples
File Paths	• Network drives • Shared internal drives
Server Names	• ODBC Connections
Credentials	• SSH Keys • Tokens (REDCap, Azure, Github, etc) • Usernames • Passwords • Blob/bucket keys
Identifiable Information	• Addresses • Names • Any PHI

Prevent Credential Leaks with Env Variables

There are a number of ways to do this. We typically use a yaml file that can be filled out with personal credentials locally. The file will not be committed to the remote repo

Create a private credentials file

The scripts use a `.yml` file that contains a list of API tokens, server names, and usernames/passwords specific to each individual user. There are two `.yml` files. One is a template (containing no actual passwords..) that exists in the repo and serves as a template so every individual user can keep up to date with new credential additions. The other is the individual **`creds.yml`** that is in the repo's `.gitignore`. This file will never exist in the repo and only exist locally (in the user's C drive).

`creds.yml` details

The `.yml` file can work with multiple programming languages including R and Python. They are read in the same way and can be easily adjusted when adding new passwords or using them as configuration files.

They look like this:

Listing 3 `local-credentials.yml`

```yaml
# Default is needed to distinguish values.
# Leave a blank line (NO SPACES) as the last line in this file or things will break
# Quotes aren't necessary, but can be used.
default:
  conn_list_wdrs:
    Driver: "SQL Server Native Client 11.0"
    Server:
    Database:
    Trusted_connection:
    ApplicationIntent:

  fulgent:
    username: <USERNAME>
    password: <PASSWORD>
```

You can have different variables assigned to unique lists, which allows for easy configuration. For example, the list starting with `default` has variables `conn_list_wdrs` and `fulgent`. You can have a different list of variables within the same file like this:

Now there is a `test` list with its own variables. This lets us switch a set of variables within our scripts. `default` applies to the main credentials where `test` can distinguish which variables should be test or dev scripts specific. Notice below that you can now call the credentials from a `.yml` file into an R or Python script and the actual credentials will never exist in the code pushed to the repo.

We can even get more specific and add an `if-else` statement to specify which credential we want to select. This can be helpful if we have a CI/CD pipeline and have a script automatically run on a task scheduler or cron job. We can call the credentials we want in the command line and have the command line code run in my task scheduler. That way we can use multiple different versions of the same script and have all of it be automated. For example, the middle panel uses the `commandArgs()` to pull any arguments passed to the script in a shell/command line script. In the right panel, the shell script has `production` and `test` as second arguments. These are passed to the R script as `arg[2]`. Now we can use `arg[2]` in the if-else statement to conditionally select credentials and do it automatically.

Safe Guards - Prevent Accidental Leaks!

Once you have the credentials.yml template in your repo, make sure that nobody on your team (or anyone with write access..) is able to accidentally push changes to the template. We don't want someone's passwords or API tokens to exist in GitHub.

Listing 4 `local-credentials.yml`

```yaml
# Default is needed to distinguish values.
# Leave a blank line (NO SPACES) as the last line in this file or things will break
# Quotes aren't necessary, but can be used.
default:
  conn_list_wdrs:
    Driver: "SQL Server Native Client 11.0"
    Server:
    Database:
    Trusted_connection:
    ApplicationIntent:

  fulgent:
    username: <USERNAME>
    password: <PASSWORD>

test:
  conn_list_wdrs:
    Driver: "SQL Server Native Client 11.0"
    Server:
    Database:
    Trusted_connection:
    ApplicationIntent:
```

This link shows how to skip any changes made to the specific file. If someone makes local changes to the template, the changes will not show in their commit. It is a safe guard.

For all individual users, run this code:

```
git update-index --skip-worktree creds_TEMPLATE.yml
```

This will tell your local git to ignore any changes made to `creds_TEMPLATE.yml`, but also allow it to exist in the repo (since `.gitignore` will prevent it from being in the repo)

If you need to update the template file run this:

```
git update-index --no-skip-worktree creds_TEMPLATE.yml
```

This will allow changes to the template. **So when you need to update the template, use this code**

And to get a list of files that are "skipped", use this code:

```
git ls-files -v . | grep ^S
```

Listing 5 `script-in-repo.R`

```r
# this script is in the repo, but credentials are hidden
library(yaml)

# read in the local credentials yaml file
creds <- yaml::read_yaml("path/to/local-credentials.yml")

# pull in the credentials
server_name <- creds$default$conn_list_wdrs$server
```

Listing 6 `script-in-repo.R`

```r
args <- commandArgs(TRUE)

# this script is in the repo, but credentials are hidden
library(yaml)

# read in the local credentials yaml file
creds <- yaml::read_yaml("path/to/local-credentials.yml")

# pull in the credentials
if(args[2] == "production"){
  server_name <- creds$default$conn_list_wdrs$server
} else if(args[2] == "test"){
  server_name <- creds$test$conn_list_wdrs$server
}
```

3 Security Guardrails

Using a `.gitignore` file for environmental variables/credentials is an excellent guardrail and promotes good coding habits, but we may also want additional guardrails such as hooks.

Hooks are processes that run in the background and can prevent code from being pushed if there is a security flaw. There are two hooks we could use for security; pre-commit hooks and pre-receive hooks

3.1 Pre-commit Hooks

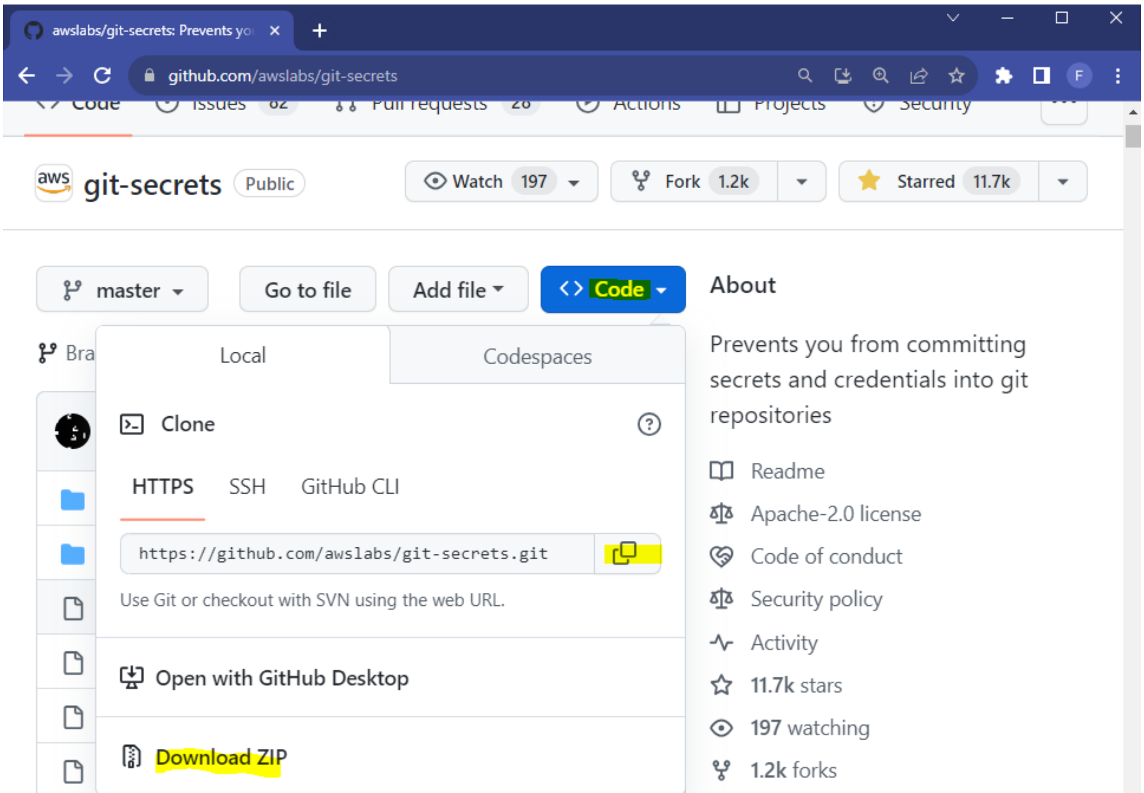
Pre-commit hooks run a process locally when the user attempts to commit code to a git branch. Hooks have many uses. Here we can use them as a security guardrail to prevent accidental credential leaks in committed code.

1. Clone or download the AWS Git Secrets repo from awslabs GitHub

Listing 7 `shell-trigger-script.sh`

```
# Run the production code
$ Rscript -e "source('path/script_in_repo.R');" production

# Run the test/dev code
$ Rscript -e "source('path/script_in_repo.R');" test
```



```
https://github.com/awslabs/git-secrets.git
```

Use Git or checkout with SVN using the web URL.

Prevents you from committing secrets and credentials into git repositories

2. Extract zip
3. Open folder and right click install.ps1

    a. Run in Power Shell
    b. Type Y to give permission

4. CD to a directory where you have the git repository you want to upload, either in PowerShell or R studio terminal ❯_

Listing 8 `PowerShell`

```
PS > cd path/to/repo/root
```

5. Run git secrets –install

Listing 9 `PowerShell`

```
git secrets --install
```

6. Copy the regex file containing the secret patterns into your folder. – make text file – discuss with team what all we want to make illegal.

7. Run git secrets –add-provider – cat ./secrets_key

Listing 10 PowerShell

```
git secrets --add-provider -- cat ./secrets_key
```

You can also add prohibited patterns like this

Listing 11 PowerShell

```
# add a pattern
git secrets --add '[A-Z0-9]{20}'

# add a literal string, the + is escaped
git secrets --add --literal 'foo+bar'

# add an allowed pattern
git secrets --add -a 'allowed pattern'
```

8. Test Git history by running

Listing 12 PowerShell

```
git secrets --scan-history
```

9. If something gets flagged and you don't care about your history anymore: Delete .git folder and reinitialize repository

   I would take caution about this point. There might be better ways to clean your git history if you don't want to get rid of everything.

10. Test on one of my projects to see if rebasing is a sustainable option
11. Make repo public
12. Will automatically scan on every commit and won't let it commit unless it's clean - Create a few files to show it working

> **Note**
>
> We can't use the "Non capture group" feature of regex. Meaning we can't use patterns like this in our regex: (?:abc) – see https://regexr.com IMPORTANT: Tab separate your regex expressions. Making new lines caused a bit of chaos and took really long to figure out. (you can use multiple tabs to separate them more visually)

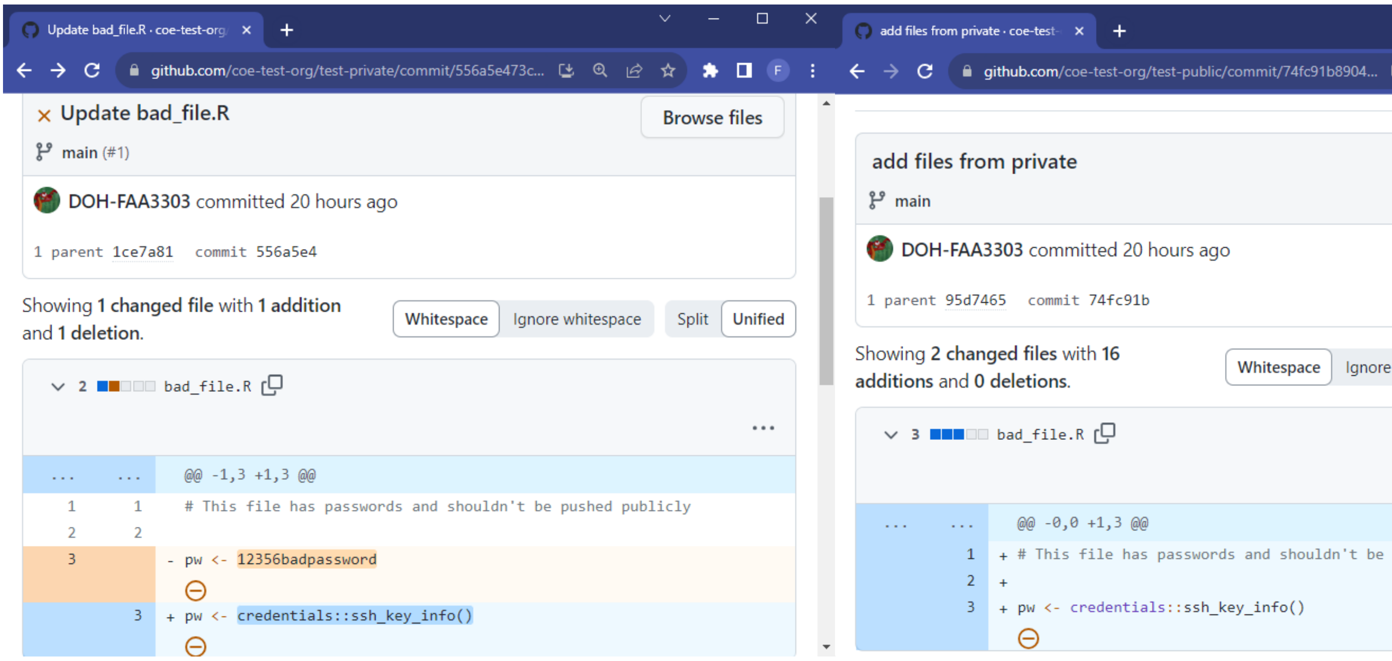
3.2 Pre-Recieve Hooks

These are still being investigated. They are remote hooks (not local like pre-commit hooks) that can be deployed throughout the Github organization. They can block certain commits from ever being pushed to the remote repo. They may make things unnecessarily complicated

3.3 Pushing Private Code to Public Repos

We may wish to take private codes and push them to a public repo. We need to make sure that the public code doesn't not contain sensitive or forbidden data/code, so cleaning up the private repo is important before pushing.

There are a few ways to do this, but the easiest way is to copy the clean private code to the public repo, that is, copy all the files you want to add publicly but **do not copy the `.git`** folder. If the private repo has a dirty git history we will not want that history in the public repo because the sensitive data will then be publicly available.



github.com/coe-test-org/test-private/commit/556a5e473c...

Update bad_file.R

main (#1)

DOH-FAA3303 committed 20 hours ago

1 parent 1ce7a81 commit 556a5e4

Showing **1 changed file** with **1 addition** and **1 deletion**.

2 bad_file.R

```
@@ -1,3 +1,3 @@
  # This file has passwords and shouldn't be pushed publicly

- pw <- 12356badpassword
+ pw <- credentials::ssh_key_info()
```

github.com/coe-test-org/test-public/commit/74fc91b8904...

add files from private

main

DOH-FAA3303 committed 20 hours ago

1 parent 95d7465 commit 74fc91b

Showing **2 changed files** with **16 additions** and **0 deletions**.

3 bad_file.R

```
@@ -0,0 +1,3 @@
+ # This file has passwords and shouldn't be
+
+ pw <- credentials::ssh_key_info()
```

! The private repository on the left still contains sensitive information in the git history. The public repository on the right has a clean git history because we copied only the current clean files from the private repo and did not attach its git history (which lives in the hidden `.git` folder)

4 Code Reviewers/Github Operations Team

With the guardrails above in place there should be few chances that credentials get pushed to a repo. However accidents may still happen. We want to make sure that anyone who opens up a repo in the Github organization adheres to the rules, has the proper credential/coding set-up, and installs their local pre-commit hooks properly.

It may be useful to have a team within the organization that helps with repo set-up. The team would help avoid a scenario where a person opens up a repo without reading this documentation and understanding the rules (and thus potentially breaking security rules).

This Github Operations Team could also be helpful in managing permissions for members in the organization. See the video below on how the company Qualcomm manages their Github organization and how they use a Github Operations Team to guide new members access/repo development

https://www.youtube.com/embed/1T4HAPBFbb0?si=YRsUYXIxLPhdr41T

5 Licensing

Summary

- Licenses prevent code theft and inappropriate redistribution of code.
- Review common open-source licenses
- License types vary depending on repo goals

6 General License Info

Below is a list of common open-source licenses.

There isn't a one size fits all license, so thankfully there are a variety of options. Here are two common ones:

7 GNU GPL licenses

a. These are the strong licenses
b. Prevents someone from taking our code and privatizing it (and making money off of it)
c. Someone can still use our code, they just need to ensure that what they're doing with it is open-source
d. "Copyright and license notices must be preserved."
e. "Contributors provide an express grant of patent rights. When a modified version is used to provide a service over a network, the complete source code of the modified version must be made available."

8 MIT license

a. I think this is the most commonly used one
b. "short and simple permissive license… only requiring preservation of copyright and license notices"
c. "Licensed works, modifications, and larger works may be distributed under different terms and without source code."
d. Someone could basically do whatever they want with the code.
e. Nextstain/ncov repo is currently using this

And here are a couple of youtube videos that were helping in explaining licensing

https://www.youtube.com/embed/rbQg9DY_4y0?si=OvU9vLBHX43dTIcA

https://www.youtube.com/embed/ndORMSnb2nw?si=tkUzjwZYWKfrLTEU

9 Policies

🔓 Objectives

- Ensure that all repos in the org have the required documents
- Set policy rules at the Organization level
- Repos need to have reproducible code
- Repos need to have documentation

In the Github Organization we may require all repositories to contain certain documents. For example, we want to make sure that every repo has a `CODE OF CONDUCT` document that is a general policy applied throughout the organization.

Here's a list of required documents:

`README`

`README` files are instructions or documentation on how to use your software. It should give a quick introduction to the repo and instructions on how to install or run the code.

`CODE_OF_CONDUCT`

A Code of Conduct can let a user know what the rules of the organization are and how any wrongful behavior will be addressed. The document will provide the "standards for how to engage in a community"

`CONTRIBUTING.md`

This file should appear in the issue tab in a repo. It lets a user know how they can contribute to the project and if they need to sign any forms before contributing. Some larger organizations require that a person knows what they are contributing to and they must sign a form acknowledging that any software/code contributions to the project will be used and cannot be retracted by the user. The code submitted may also be used to develop processes but the organization will not pay the individual contributor (since this is open-source, we only look for open-source contributions)

`LICENSE`

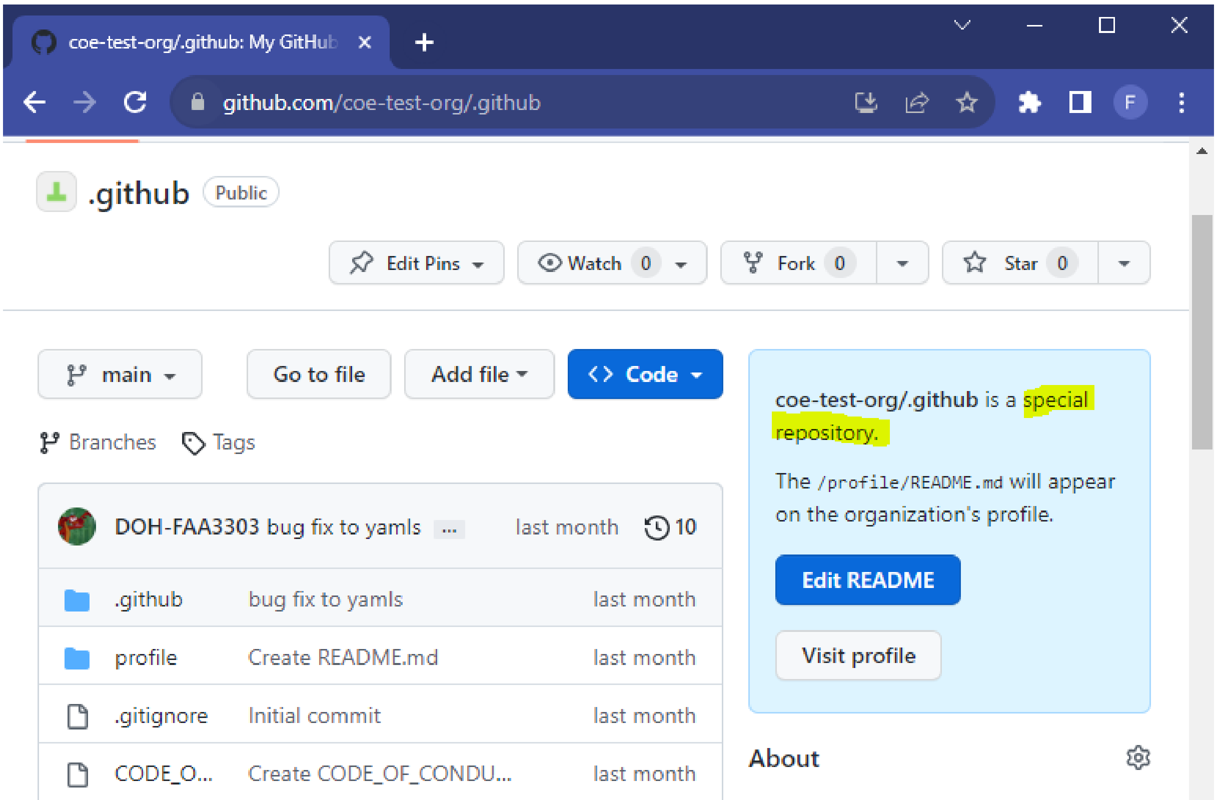
These should be and are set at the repo level. There will be many different licenses to choose from that will depend on the specific repo. More on that here.

10 Set Policy Rules at Org Level

Policy rules may include requiring certain documents in each repo or requiring that a person sign every commit.

10.1 Document Requirements with `.github` Repos

You can set most policy rules and create documents for each repo at the organization level by using a special `.github` repo. Dot files and dot folders have special functionality in some software. For Github, the `.github` folder defines workflows for things like Github Actions in a repo. A `.github` *repository* on the other hand defines *organization level* rules and templates.



coe-test-org/.github is a special repository.

The /profile/README.md will appear on the organization's profile.

DOH-FAA3303 bug fix to yamls ... last month 10

.github	bug fix to yamls	last month
profile	Create README.md	last month
.gitignore	Initial commit	last month
CODE_O...	Create CODE_OF_CONDU...	last month

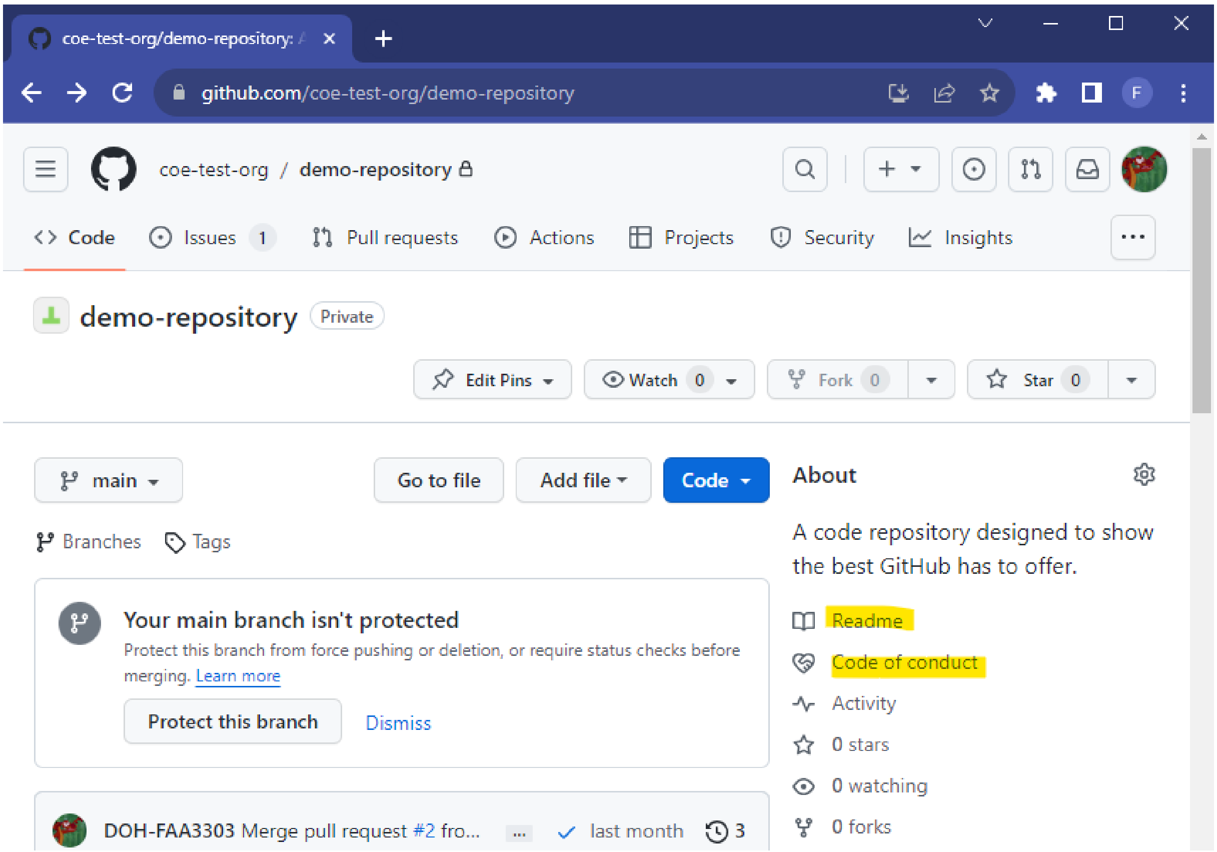
In order to write and set these policies at the organization level we can put them at the root of the `.github` repository and edit them there.

Listing 13 `PowerShell`

```
$ tree /f
C:.
    .gitignore
    CODE_OF_CONDUCT.md
    CONTRIBUTING.md
    LICENSE
    README.md

   .github
   profile
           README.md
```

Take a look above. I have the required documents/policies at the root of the `.github` repo directory. Now if I open up any given repo in the organization I will find a link to those files:



A code repository designed to show the best GitHub has to offer.

Your main branch isn't protected

Protect this branch from force pushing or deletion, or require status checks before merging. Learn more

Protect this branch
Dismiss

If you click on the `CODE_OF_CONDUCT` link it will take you right to the `.github` repo and open the `CODE_OF_CONDUCT.md` file there:

Now you can set organization level policies from the `.github` repo and they will automatically populate in *all* existing and new repositories *unless there are repo specific policies in place.* If a repo already has its own policies they will not be overwritten.

11 Set Templates at the Org Level

Aside from policy documents, you can make templates at the organization level. Two commonly used templates are issue templates and discussion templates.

In the public repos there may be end users that may have limited experience using Github. If they want to submit an issue or ask a question they get lost. Templates can help them form a question or idea. Templates can also help standardize how issues and discussions are maintained throughout the organization.

Structuring the format of issues and discussions can make the author and the end-user's lives easier.

In the `.github` repo I made a *folder* called `.github`. This is a special folder that can hold Github Action workflows and more, as mentioned above.

In the `.github` *folder* I have a folder called `DISCUSSION_TEMPLATE` and another called `ISSUE_TEMPLATE`. These are special folders that Github recognizes as discussion and issue folders that will set templates at the repo (or in this case the org) level.

Each Folder has `.yml` files in it that are basically Github instructions on how to format issues and discussions.

Listing 14 PowerShell

```
$ tree /f
C:.
    .gitignore
    CODE_OF_CONDUCT.md
    CONTRIBUTING.md
    LICENSE
    README.md

  .github
       pull_request_template.md

      DISCUSSION_TEMPLATE
           feature-requests.yml
           q-a.yml
           show-and-tell.yml

      ISSUE_TEMPLATE
            bug_report.yml
            config.yml
            feature_request.yml

  profile
        README.md
```

For example, in the `ISSUE_TEMPLATE` folder I have a `.yml` file called `bug_report.yml`. This file contains the structure for how someone can report a bug.

Now, when someone clicks on the `Issues` tab in a repo in this organization they will be met with the `Bug Report` template:

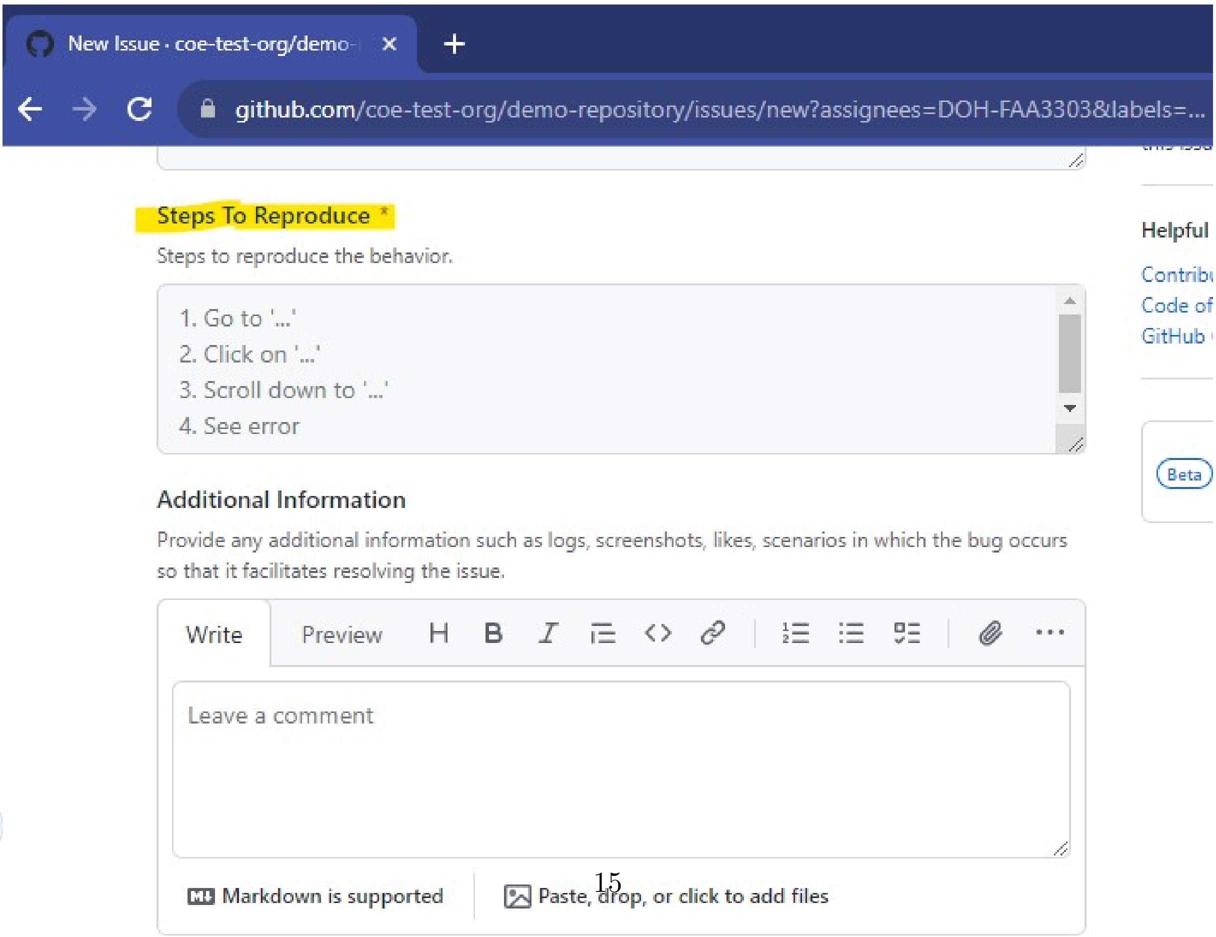
Notice that in the template you can create text areas and pre-fill those areas with suggestions. You can even require that someone fills out those areas before they can submit the issue:

Listing 15 bug-report.yml

```yaml
name: Bug Report
description: File a bug report here
title: "[BUG]: "
labels: ["bug"]
assignees: ["DOH-FAA3303"]
body:
  type: markdown
    attributes:
      value: |
        Thanks for taking the time to fill out this bug report
        Make sure there aren't any open/closed issues for this topic
```

Listing 16 bug-report.yml

```yaml
- type: textarea
    id: steps-to-reproduce
    attributes:
      label: Steps To Reproduce
      description: Steps to reproduce the behavior.
      placeholder: |
        1. Go to '...'
        2. Click on '...'
        3. Scroll down to '...'
        4. See error
    validations:
      required: true
```



New Issue · coe-test-org/demo-

github.com/coe-test-org/demo-repository/issues/new?assignees=DOH-FAA3303&labels=...

Steps To Reproduce *

Steps to reproduce the behavior.

1. Go to '...'
2. Click on '...'
3. Scroll down to '...'
4. See error

Additional Information

Provide any additional information such as logs, screenshots, likes, scenarios in which the bug occurs so that it facilitates resolving the issue.

Leave a comment

11.1 Commit Sign-Off Requirement - Github Apps

We may want to require authors or reviewers to sign-off on commits to a repo. This is sometimes established in projects to "ensure that copyrighted code not released under an appropriate free software (open source) license is not included in the kernel."

You can install a Github App in the organization and it will be applied to all repos. The DCO App (Developer Certificate of Origin) is popular and lightweight. To install it in the organization, click on Configure and it will give you the option to configure it with the organization of choice.

12 IaC

Infrastructure as Code (IaC) can be helpful when managing administration tasks or writing hooks at the org level.