EPMO CMD Standard VA GitHub Repository Deployment

Contents

[Where to Start 3](#_Toc17102933)

[Defining Repository Type 3](#_Toc17102934)

[Naming Convention: GitHub Teams 3](#_Toc17102935)

[Creating Teams 4](#_Toc17102936)

[VA Git Repositories 5](#_Toc17102937)

[Git Repository Structures 5](#_Toc17102938)

[Naming Convention: GitHub Repository 6](#_Toc17102939)

[Creating a New Repository 6](#_Toc17102940)

[Configuring Repository Settings 7](#_Toc17102941)

[Repository Options 7](#_Toc17102942)

[Collaborators & teams 8](#_Toc17102943)

[Branching 9](#_Toc17102944)

[Branch strategy 9](#_Toc17102945)

[Creating a Branch 9](#_Toc17102946)

[Branch Rules 10](#_Toc17102947)

[Master Branch Rules 10](#_Toc17102948)

[Branch Rules 10](#_Toc17102949)

**Document Revision History**

| Date | Document Version | Description | Author | VA Group or Contract Company |
| --- | --- | --- | --- | --- |
| 08/16/2019 | 1.0 | Initial Version | S.Souza | EPMO CMPSS |

Deploying VA GitHub Repository Procedures

The document is arranged to facilitate the administrator’s tasks with creating a Veterans Administration as per the standards for deploying a GitHub repository in the GitHub organizations: “Github.com/**department-of-veterans-affairs**” or the “https://github.ec.va.gov/**EPMO**”. Most importantly this document establishes several naming conventions for teams, repositories and branches.

## Where to Start

New implementations of repositories will be in GitHub.com, unless the project has a justification due to sensitive data within the development files that pose a risk of exposure/release to the public. The code cannot have sensitive data or PII/PHI if it is hosted in GitHub.com. This data is different from the runtime application that may manage or use sensitive data. The runtime data does not determine where the development artifacts are managed.

\* Note: **At this time all VistA development will be hosted in the Rational repositories until VistA is migrated to GitHub.ec.va.gov**

If your application has VistA code and non-VistA code such as Delphi or Java wrappers, please work with your GitHub implementation team for proper GitHub standup.

## Defining Repositories Type

In the VA, when defining repositories there are three general type; **product,** **code-project** and **infrastructure-only**. Each is supported by its respective repository template; “**va-code-project-template**” and “**va-product-documentation-template**”.

A project’s initial requirement will be to define its designated repository type, as defined below:

**product**: Each product will have one product repository.

The product repository will have a **README.MD** file used to track **code-project** repositories that makeup the product. The product repository will be used for release readiness self-certification, agile planning, Epics, Stories, Defects, Tasks, Tests, Risk, documentation and change management in conjunction with the code project change management processes, if no other tools are integrated with GitHub. There are many tools and integrations that are possible, and the use of the product repository will be adapted as needed depending on the integrations and the capabilities of the tools being integrated.

**code-project**: Each software product will have at least one code-project repository. If only one repository is required, then use “code” or “application” for <code-project> for example cpe-code

The code-project should not be one large repository that is used to source control the entire product. The code-project should reflect how the developer works including organization within the IDE as projects/packages/solutions and how those are built, tested and prepared for deployment. The code-project repositories should/could:

• Independently be managed (repository setting, build files, config files, branching and branching rules etc.)

• Represent the logical division of the software into obvious functional elements of the software architecture

• Provide the realization of a set of interfaces

• Be a non-trivial, nearly independent, and replaceable part of a product that fulfills a clear function.

• Be a modular, deployable, and replaceable part of the product

• Typically specified by one or more classifiers (e.g., implementation classes) that reside on it

• Be built independently (although it may have dependencies or be a dependency at runtime)

## Naming Convention: GitHub Teams

The following are standard Git Team types for VA Git repositories.

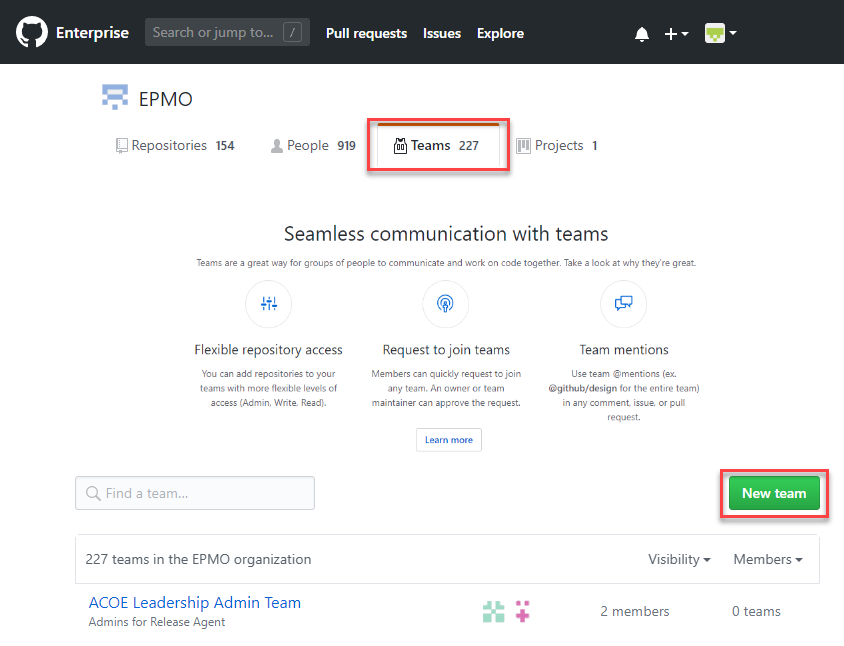
Table 1 - VA Git Repository Team Types

| Team Name Format | Example | Definition |
| --- | --- | --- |
| <**product**>-team | crt-team | Membership within the product Team consists of all users who need access to the repository. |
| <**product**>-developers | crt-developers | Membership within the developer Team consists of developers who need write access to the repository. |
| <**product**>-devlead | crt-devlead | Membership within the devlead Team consists of the developer team lead and a designated backup person who are responsible for accepting pull requests and merging code into the dev branch for the repository. |
| <**product**>-sqa-lead | crt-sqa-lead | Membership within the sqa-lead Team consists of the sqa lead and designated backup person who are responsible for accepting pull requests and merging code into the corresponding sqa branch within the repository structure.  **Note:** Git sqa branches correspond with a product’s SQA environment(s). Depending on product, VA Git repositories may have more than one sqa branch. |
| <**product**>-preprod-lead | crt-preprod-lead | Membership within the preprod-lead Team consists of the lead and designated backup person who are responsible for accepting pull requests and merging code into the preprod branch within the repository. |
| <**product**>-prod-owner | crt-prod-owner | Membership within the prod-owner Team consists of the product owner and designated backup person who are responsible for accepting pull requests and merging code into the master branch within the repository. |
| <**product**>-admin | crt-admin | Product maintainers with Admin permissions to read, clone, push and add collaborators to the repository. |

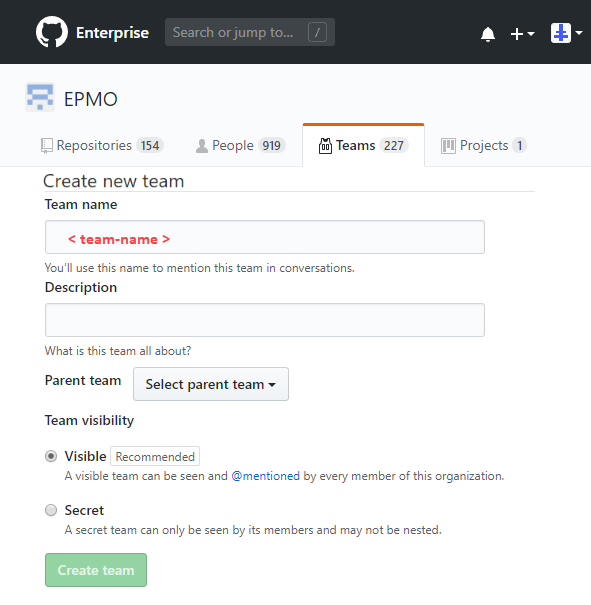
## Creating Teams

Prior to configuring the repository setting it is necessary to create the teams and their membership.

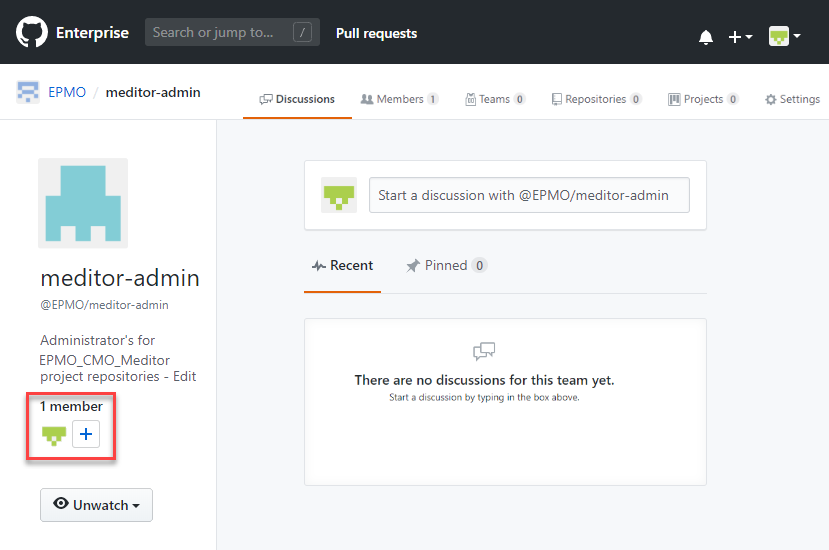
1. Navigate to your organization’s landing page for EPMO it is <https://github.ec.va.gov/EPMO> or https://github.com/department-of-veterans-affairs
2. Select the Team tab



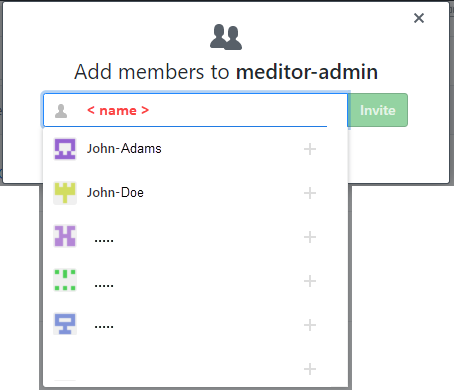
1. GitHub opens the New Team wizard
2. Following the standard naming convention enter the team name
3. Enter a brief description
4. Select **Visible**
5. Select **Create team**



1. Git Hub verifies the teams creation, the user can then add a new member by selecting the plus ( + ) in the left pane



1. GitHub prompts the user with a search dialogue window.
2. Enter all or part of a name, GitHub dynamically presents a list in the dropdown
3. Select the user from the available members in the list result



**Note: For those repositories created** **in** <https://github.ec.va.gov/EPMO>**, add user VHAISP-RTCGitHubUsr, Jeff.Simon to the team “**<**product**>-team” for RTC plugin/integration with GitHub

## VA Git Repositories

This document has been written as a quick-reference guide for Enterprise Program Management Office (EPMO) Configuration Management Department (CMD) Configuration Managers when advising their projects.

## Git Repository Structures

The following are recommended Git repository structures by type of VA project team or VA product. Depending on team or product, further structure customizations may be needed.

**Note:** Git repositories are similar in function to Rational components—they house or contain source artifact files and folders. However, baselining will work differently in Git repositories.

| Team or Product | Example | Recommended Structure | Notes |
| --- | --- | --- | --- |
| Service Team | * Release Readiness Office * EPMO CMD | * Documentation | Service teams tend to source control documentation artifacts only. |
| Product | * BIO BIS * EPRS * VDIF | * Documentation * Source | Repositories based on code base and architecture. |
| Infrastructure-only Product | * Example1 * Example2 | * Documentation * Infrastructure-specific artifacts | Non-documentation repository(-ies) based on what is being delivered. |
| Other | * Example1 * Example2 | * Documentation | Repositories based on code base and architecture. |

## Naming Convention: GitHub Repository

The names of the repository will be in all lower case and spaces will be substituted with dashes ( “-“ ). The syntax or repository name pattern is as follows:

Product repository <**product**>-**product**

Code-project repository <**product**>-<component or if there is only one “code” or “application”>

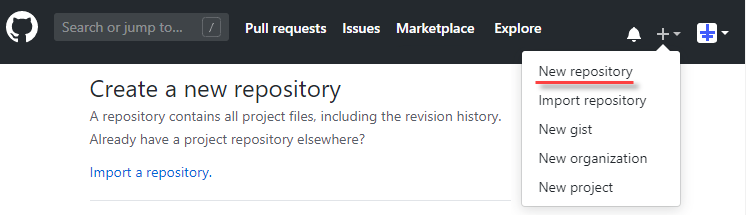
Infrastructure-only repository <**product**>-<component or if there is only one “code”>

This is not an exhaustive list and projects may have a combination of the above.

## Creating a New Repository

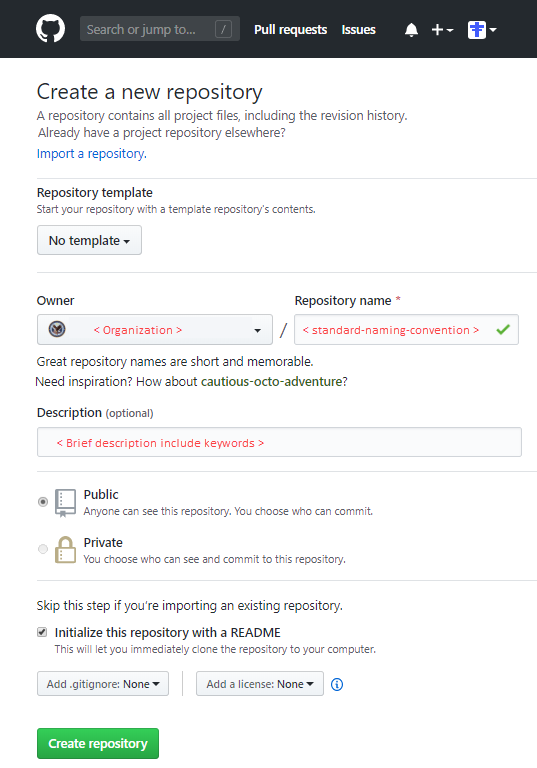
Open to the EPMO VA Enterprise GitHub <https://github.ec.va.gov/EPMO> or https://github.com/department-of-veterans-affairs

To create a new GitHub repository, in the upper right corner next to your profile icon select the plus button and **New repository**.



The "Create a new repository" wizard opens. Complete or select the following:

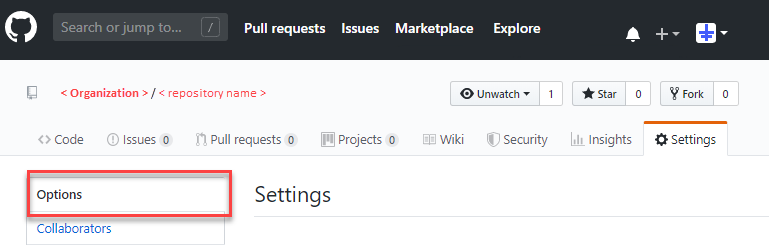
1. Repository template: \*Note this feature is not available in <https://github.ec.va.gov>
   1. No Template
   2. **department-of-veterans-affairs/va-code-project-template**
   3. **department-of-veterans-affairs/va-product-documentation-template**
2. Owner:
   1. **EPMO**
   2. “**department-of-veterans-affairs**”
   3. If applicable other corresponding va organization. Just don't make yourself the owner
3. Repository name = see naming standard document
4. Description = recommend a short (optional) description and ideally a tag which makes this repository and all related repository searchable (include the product acronym).
5. For <https://github.ec.va.gov/> Select **Public** so anyone can see this repository. You choose who can commit.  
   For <https://github.com/department-of-veterans-affairs> Select **Private** You choose who can see and commit …
6. Check Initialize this repository with a README. This will let you immediately clone the repository to your computer.
7. Leave "Add .gitignore **None**"
8. Leave "Add a license **None**"
9. Select Create repository



## Configuring Repository Settings

As per the VA standards the GitHub repository requires additional configuration.

1. Select the **Settings** tab of the repository’s main header bar



### Repository’s Options

Select the **Options** section from the left pane

Select the following Features

1. Wikis
2. Restrict editing to users in teams with push access only
3. Issues
4. Projects

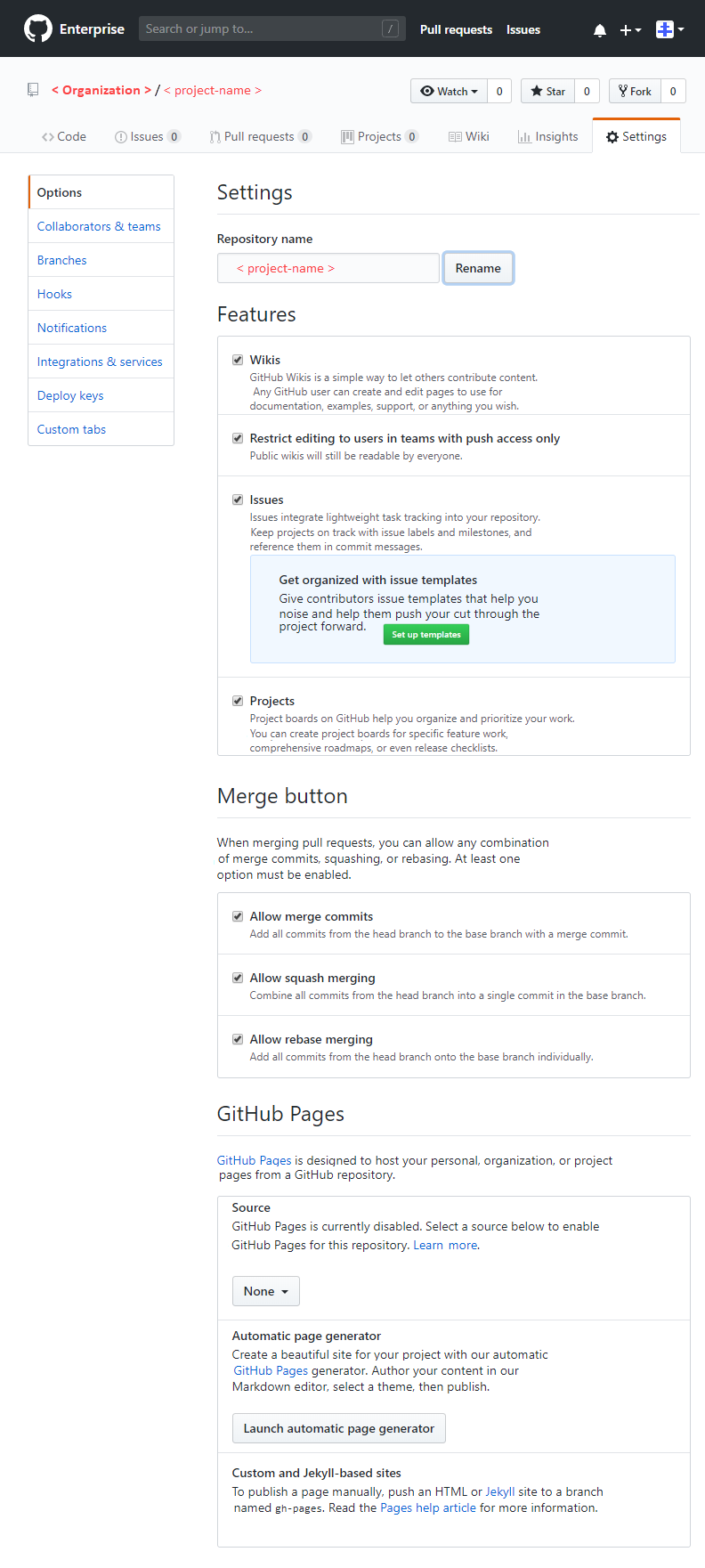
Select the following Merge Buttons

1. Allow merge commits
2. Allow squash merging
3. Allow rebase merging

GitHub Pages

1. Source **None**
2. Automatic page generator **Launch automatic page generator**

**Caution: Option only available in** <https://github.ec.va.gov/EPMO> **This repository is private but the published site will be public. GitHub pages is currently disabled. Select a source below to enable GitHub Pages for this repository.** [Learn more](https://help.github.com/articles/configuring-a-publishing-source-for-github-pages/)

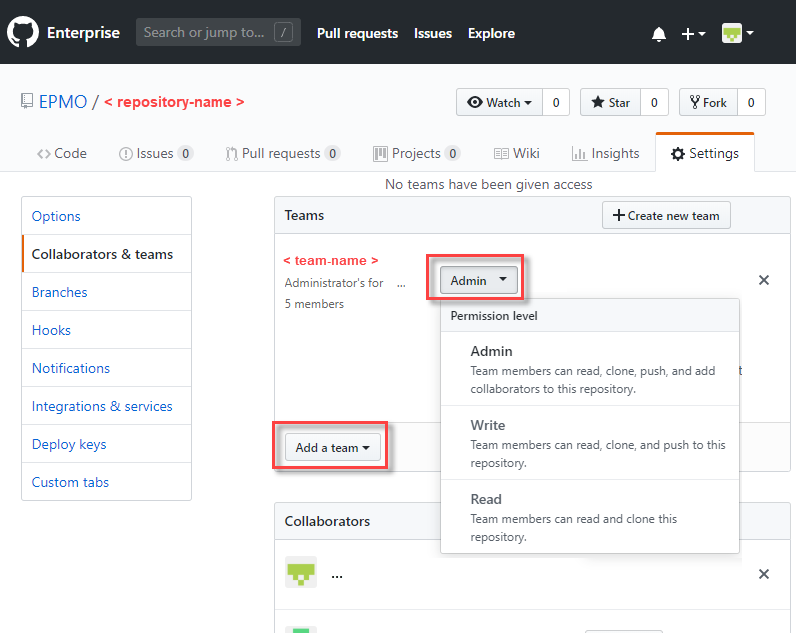


### Collaborators & teams

The teams created must now be associated to the specific repository with designated permissions.

1. Select the **Collaborators & teams** section from the left pane
2. Select **Add** a team and type in the name of the team (partial search results are displayed to pick from)
3. Once added the permission level buttons will display
4. Select either **Admin**, **Write** or **Read**. (Teams with “Read” permission cannot be associated to a branch rule)

Repeat for every associated team created for the repository



## Branching

From the repository settings screen the user can create an manage the master and subsequent branches. The VA has implemented a basis for branching strategy that most repositories will need to deploy.

### Branch strategy

#### **product repository**

Standard recommendation for branching a product repository is shown below.

**master** (aka Prod)

Branch: <**product**>-<**project/program**>

Branch: < collaborative branch name has no standards >-< **work item #/issue#** >

#### **code-project repository**

Standard recommendation for branching code project repository is shown below. The depth depends on the number of users collaborating on the branch and the number of environments used in the development process.

**master** (aka Prod)

Branch: <**product**>-**preprod-env**

Branch: <**product**>-sqa***(n)*-env  
 \* Note: *(n)*** indicated a number variable for differentiating multiple sqa environment’s branch

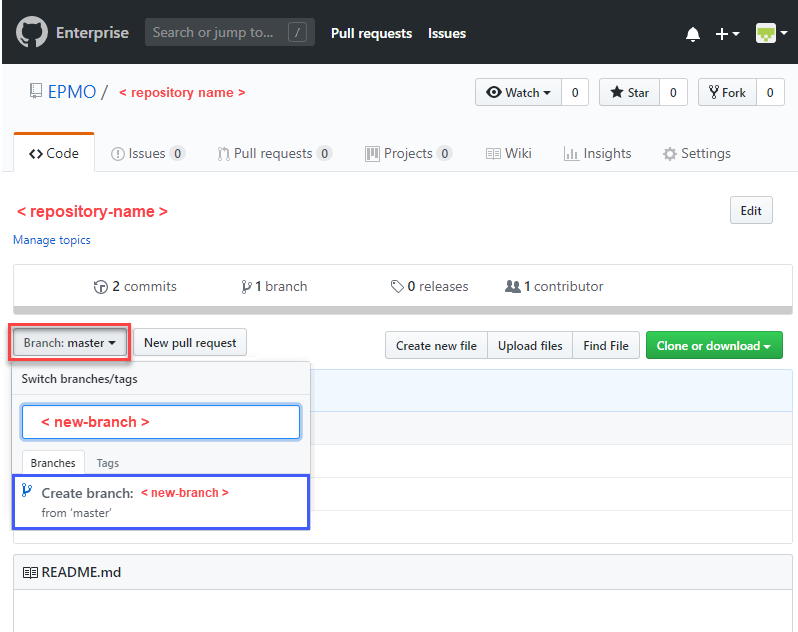
Branch: <**product**>-**dev-env**  
(each devlead will be responsible for issuing the pull   
Request to the appropriate SQA env branch)

Branch: <**product**>-< **developer-name** >-< **work item #/issue#** >

Branch: <**product**>-< **developer-name** >-< **work item #/issue#** >

### Creating a Branch

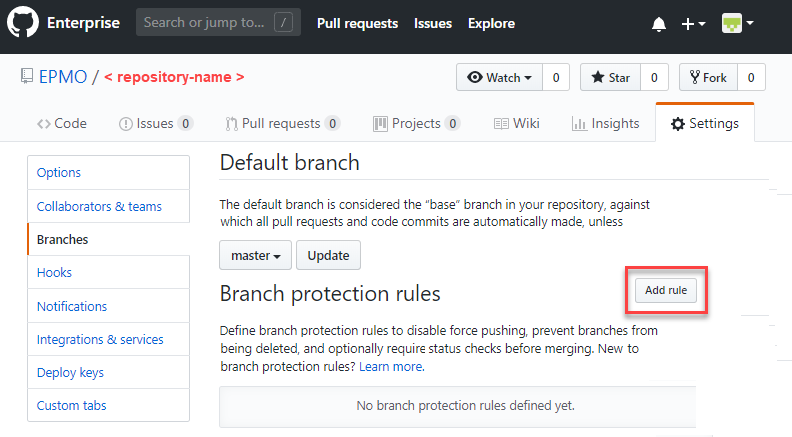
1. Select the **Code** tab of the repository’s main header bar
2. Select the origin/target branch of the pull request, select the   
   **Branch:<branch-name>** button and choose the applicable branch
3. Select **Branch:<branch-name>** a second time, type in the field which reads “Find or create a branch” in grey the branch names as per the standard naming convention.
4. Select the “Create branch: < new branch >



### Branch Rules

Returning to “Configuring Repository Settings”.

1. Select the **Settings** tab of the repository’s main header bar
2. Select the **Branches** section from the left pane
3. The first rule to set is for the master “base” branch, select master



### Master Branch Rules

For the master branch

1. In the field “Branch name pattern”, enter the branch name the rule is to be associate to, **master**
2. Select **Require pull request reviews before merging**
3. Select **Required approving reviews:**” button and choose **2**
4. Select **Dismiss stale pull request approvals when new commits are pushed**
5. Select **Require status checks to pass before merging**
6. Select **Require branches to be up to date before merging**
7. **Do not select** Require signed commits
8. Select **Include administrators**
9. Select **Restrict who can push to matching branches**
10. Search for people or teams to apply this constraint
11. Select Create

### Branch Rules

Returning to “Configuring Repository Settings”.

1. Select the **Settings** tab of the repository’s main header bar
2. Select the **Branches** section from the left pane
3. The following rules are set for each subsequent child branch, select to (i,e. preprod, sqa, dev)

\* Note: The “base” branch in your repository, against which all pull requests and code commits are automatically made, unless you specify a different branch.

For all other standard Branches

1. In the field “Branch name pattern”, enter the branch name the rule is to be associate
2. Select all the same settings as the master branch rule **except for**:
   1. Select **Required approving reviews:**” button and choose **1**
3. Search for people or teams to apply this constraint
4. Select Create

