# **GitHub Runbook**

This runbook provides a step-by-step guide for common GitHub workflows, incorporating industry-recommended best practices and security considerations, from team management to code changes and pull requests.

## **1. Teams**

Teams in GitHub are a way to organize groups of users within an organization and manage their access to repositories.

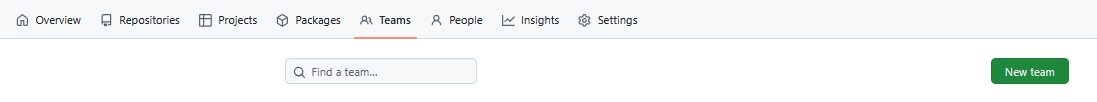
Video provides a powerful way to help you prove your point. When you click Online Video, you can paste in the embed code for the video you want to add. You can also type a keyword to search online for the video that best fits your document.

Video provides a powerful way to help you prove your point. When you click Online Video, you can paste in the embed code for the video you want to add. You can also type a keyword to search online for the video that best fits your document.

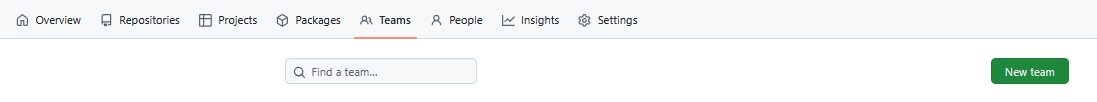
### **1.0 How to Create a Team**

Creating a team helps centralize permission management and fosters collaboration.

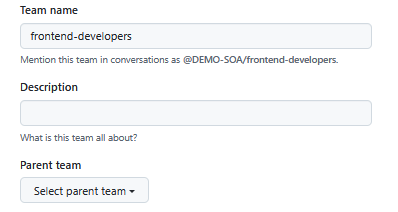
1. Navigate to your organization's page on GitHub.
2. In the Navigation Menu. click on **"Teams"**.



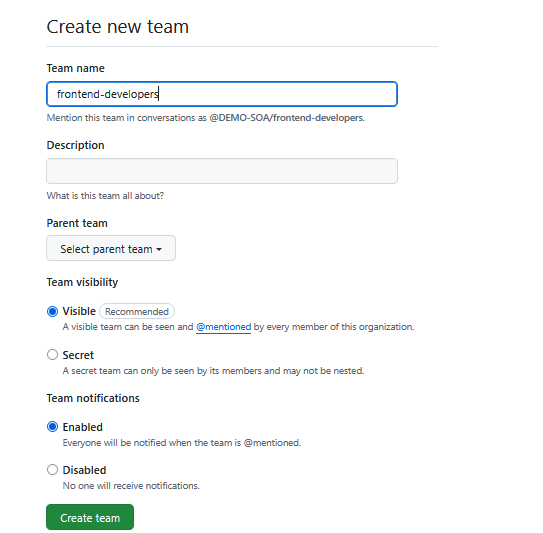
1. Click the **"New team"** button.



1. Enter a **Team name** (e.g., frontend-developers, backend-api-team).
2. Optionally, add a **Description** to describe the purpose for this team
3. Choose the **Parent team** if you want to the new team to be a subteam of another team



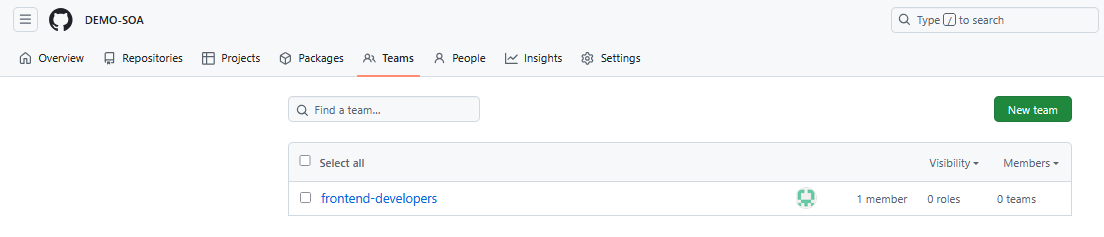
1. Select the **Visibility** (e.g Visible for all organization members, Secret for only team members).
2. Click **"Create team"**.



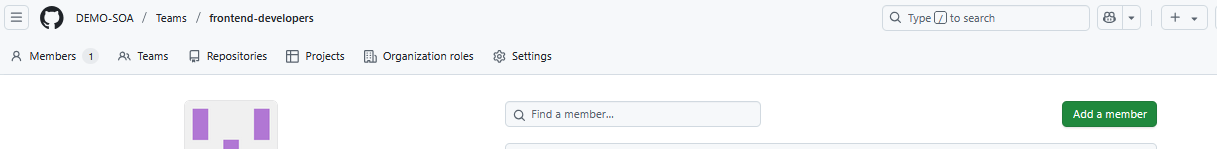
### **1.1 Add Members to a Team**

To add members to an existing team:

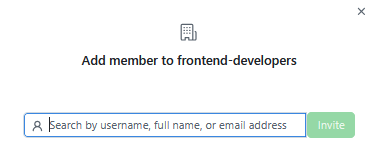
1. Navigate to your organization's page on GitHub.
2. Click on the **"Teams"** tab.
3. Select the team you wish to add members to. (e.g **frontend-developers)**



1. Click on the **"Members"** tab within the team's page.
2. Click the **"Add a member"** button.



1. Search for the GitHub username(s) of the person you want to add.
2. Select the user(s) and click **"Add [username] to team"**.



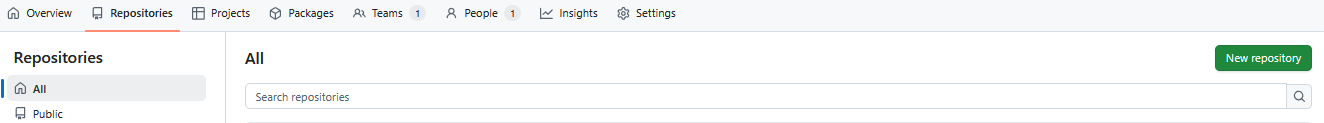
## **2. Repositories (Repos)**

A repository is the most basic element of GitHub. It's where you store all your project's files, including code, documentation, and revision history.

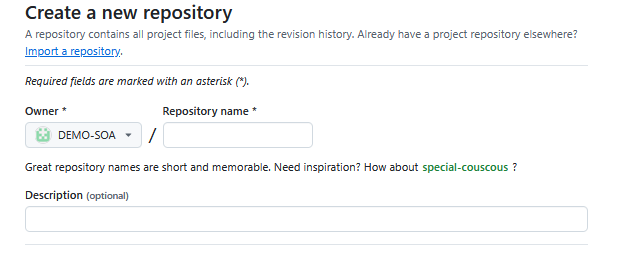
### **2.0 How to Create a Repo**

Creating a new repository is the first step for any new project on GitHub.

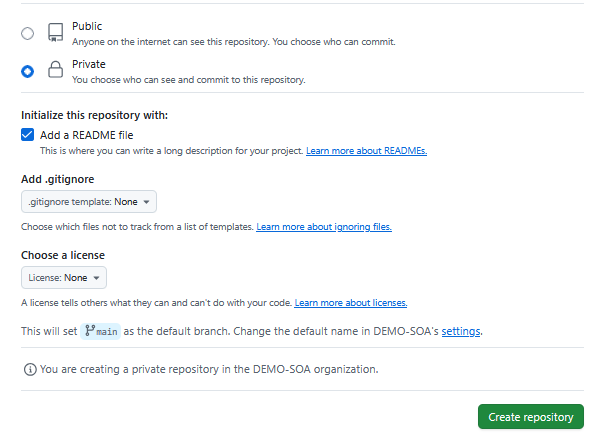
1. Navigate to your GitHub organization's page. And Click on the **Repositories** tab



1. Select **"New repository"**
2. Enter a **Repository name** (Enter your preferred Repo Name in the Box,). Make sure the **Owner** Is the name of the Organization you want the repository to be created to.
3. Optionally, add a **Description**. (To describe the objective of this repository)



1. Choose **Public** or **Private/Internal** visibility.
   * **Private (Recommended)**: You decide who can see and commit to the repository
   * **Internal:** Private to the organization, but shared across all members.
   * **Public:** Anyone on the internet can see the repository



1. Initialize the repository with:
   * **Add a README file:** Recommended for project description and setup instructions.
2. Click **"Create repository"**.

### **2.1 Role/Permission Access to a Repo**

GitHub offers various permission levels (roles) for repositories, adhering to the **principle of least privilege** is a key security best practice. Grant users only the minimum access necessary for their role.

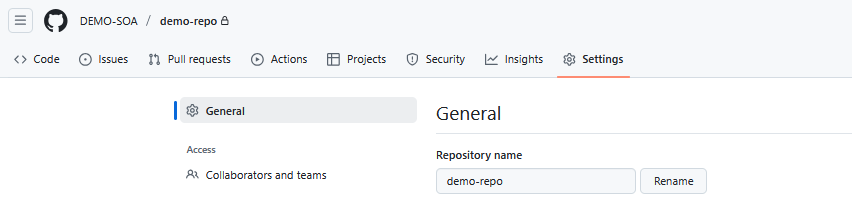
1. **Read:** Recommended for non-code contributors who want to view or discuss your project. Ideal for stakeholders or new contributors who only need to observe.
2. **Triage:** Recommended for contributors who need to manage issues and pull requests without write access.
3. **Write:** Can push to the repository, create branches, and manage pull requests and issues. This is the standard role for active developers.
4. **Maintain:** Recommended for project managers who need to manage the repository without access to sensitive or destructive actions.
5. **Admin:** Recommended for people who need full access to the project, including sensitive and destructive actions like managing security or deleting a repository.

### **2.2 Give Team Access to a Repo**

Granting a team access to a repository simplifies permission management for multiple users.

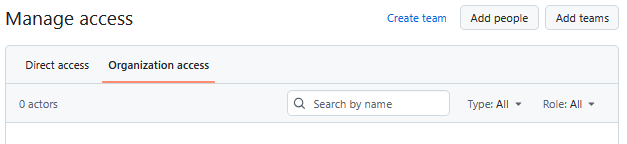
When adding teams or individuals, select the role that aligns with their responsibilities. Regularly review permissions to ensure they are still appropriate.

1. Navigate to the repository you want to manage.
2. Click on the **"Settings"** tab.
3. In the left sidebar, click on **"Collaborators and teams"**.
4. Click the **"Add team"** button.
5. Search for the team you want to add.
6. Select the team and choose the desired permission level (e.g., Read, Triage, Write, Maintain, Admin).

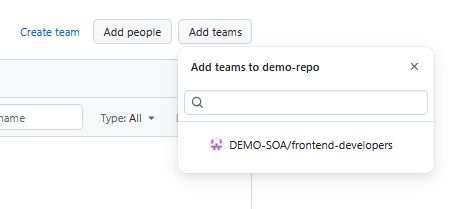


**Note:** You will be prompted to enter your GitHub Account Password to proceed, ***Enter your Password***

1. Click on Organization Access tab and Click **"Add teams"**.



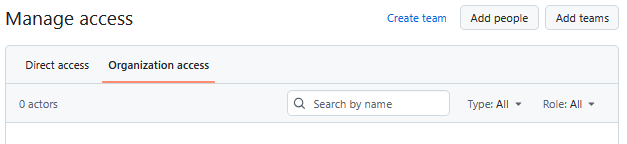
1. Search for the name of the team you want to give access to the repo



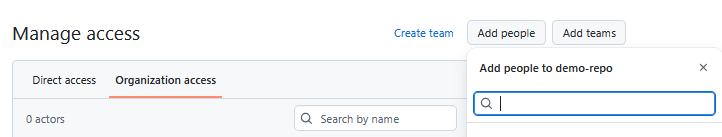
### **2.3 Give Developers Access to a Repo**

While teams are preferred for managing groups, you can also give individual developers direct access, though this is less scalable for larger teams.

1. Navigate to the repository you want to manage.
2. Click on the **"Settings"** tab.
3. In the left sidebar, click on **"Collaborators and teams"**.
4. Click the **"Add people"** button.



1. Search for the GitHub username of the **developer** you want to add.



1. Select the user and choose the desired permission level. **(Read, Write, Admin)**
2. Click **"Add [username] to repository"**. The developer will receive an invitation to join via his email.

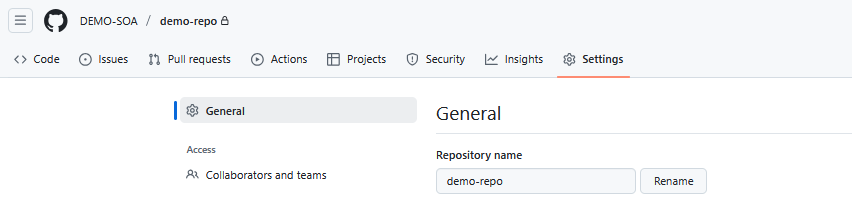
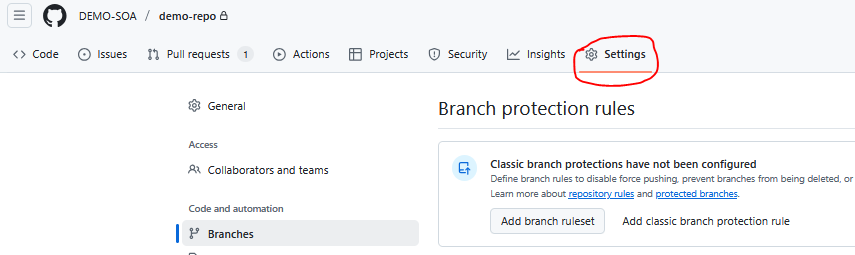
## **3. Branch Protections and Best Practices**

Branch protection rules enforce certain workflows for one or more branches, helping to maintain code quality, stability, and security. They are critical for preventing accidental or unauthorized changes to important branches like main or production.

### **3.1 Classic Branch Protection Rules**

GitHub offers two ways to configure branch protections:

* **Classic Branch Protection Rules (Repository-level):**
  + Configured directly within a specific repository's settings.
  + Each rule applies to a single branch pattern (e.g., main, feature/\*).
  + Simpler for individual repositories
  + Located under **Repository Settings > Branches**.

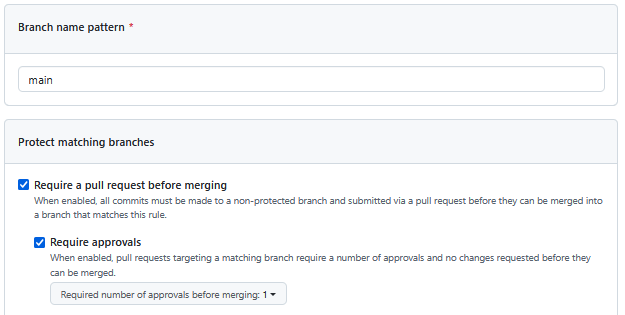


### 

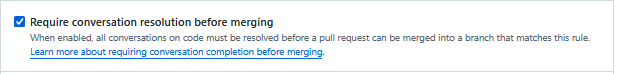
### 

### **3.2 How to Configure Branch Protections (Classic)**

1. Navigate to your repository on GitHub.
2. Click on the **"Settings"** tab.
3. In the left sidebar, click on **"Branches"**.
4. Under "Branch protection rules," click **"Add rule branch protection rule"**.
5. Enter the **Branch name pattern** (e.g., main, develop, release/\*).
6. **Configure the desired rules (Best Practices for Security & Quality):**
   * **Require a pull request before merging:**
     + **Require approvals:** Specify the number of approving reviews needed (e.g., 1 or 2). **Highly Recommended.**
     + **Require review from code owners:** If a CODEOWNERS file is present, ensures specific teams/individuals review relevant code. **Recommended.**
     + **Dismiss stale pull request approvals when new commits are pushed:** Ensures reviews are up-to-date with the latest code. **Highly Recommended.**



* + Select **Require conversation resolution before merging:** all conversations on code must be resolved before a pull request can be merged into the **main** branch



* + Select **Require linear history:** Prevents merge commits, enforcing a rebase-only workflow. This keeps the commit history clean. **Recommended for cleaner history.**

****

* + Select **Do not allow bypassing the above settings:** This settings will apply to administrators and custom roles with the "bypass branch protections" permission.

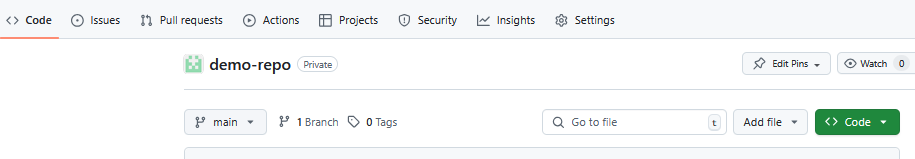


1. Click **"Create"** or **"Save changes"**.

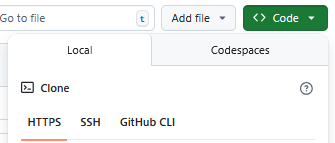
## **4. Cloning a Repo**

Cloning a repository downloads a copy of the repository from GitHub to your local machine.

1. Navigate to the repository on GitHub.
2. Click the green **"Code"** button.



1. Choose your preferred method and copy the URL.
   * **HTTPS:** https://URL/org-name/repo-name.git (Easier for initial setup, but requires re-entering credentials or using a credential helper).
   * **SSH:** git@URL:DEMO-SOA/repo-name.git (Recommended **Best Practice** for security and convenience in automated scripts or frequent pushes, as it uses SSH keys for authentication).
   * **GitHub CLI:** gh repo clone username/repo-name (If you have GitHub CLI installed).



1. Open your terminal or command prompt.
2. Navigate to the directory where you want to store the repository.
3. Run the command: git clone RepoURL
   * Example: git clone https://github.com/my-org/my-project.git
   * Example: git clone git@github.com:my-org/my-project.git

## **5. Branching**

Branches allow you to develop features, fix bugs, or experiment with new ideas in a contained area of your repository without affecting the main codebase. This is a core part of collaborative development.

### **5.0 How to Create Branches Locally**

1. Ensure you are in the local repository directory in your terminal.

Switch to the main (or master) branch and pull the latest changes to ensure your new branch is based on the most up-to-date code:  
git checkout main

git pull origin main

1. Create a new branch:  
   git checkout -b ticket-no/ticket-title  
   git checkout -b BR-0421/UPDATE-Report
2. **Best Practice:** Use descriptive branch names that indicate the purpose (e.g., ticket-no/ticket-title, feature/add-user-auth, bugfix/fix-login-error, hotfix/critical-security-patch, chore/update-dependencies). Avoid generic names like my-branch.

## **6. Making Changes Locally and Pushing to GitHub**

After creating a branch, you can make changes to the files and push them to your branch on GitHub.  
  
**Git Areas**1. Working Directory (Working Tree): This is where you edit files in your project.

Files here can be:

1. Untracked: Not yet added to Git.

2. Modified: Changed since the last commit:

Always run git status To check the status of files and the current git area

2. **Staging Area**

**Staging Area:** is an intermediary zone before making any commits

Files added here using git add are prepared for the next commit.

git add . (Adds every changes to the staging area and ready for commit)

git add filename **(**Adds file to the staging area and ready for commit)

git add foldername **(**Adds a folder to the staging area and ready for commit)

git status **(**Check to see if your changes are added/tracked in the Staging area)

3. **Local Repository**

Where commits are saved locally on your machine.

You use **git commit** to move changes from **Staging** to **Local Repo**

git commit -m "commit message"

When you're working with code and making changes, you write a short **message** (called a *commit message*) to explain **what you did**

**Best Practice:** Use conventional commit messages  
To keep things organized and clear, it's a **best practice** to **start these messages with a specific keyword**, like:

feat: — when you're **adding a new feature**fix: — when you're **fixing a bug**docs: — when you're **updating documentation**style: — when you're **cleaning up code formatting** (like spaces or commas)

4. **Remote RepositoryThe shared repo** hosted on GitHub

Collaborators push to and pull from this repo. Sync changes using:

git branch (To check the branches and the active branch, (e.g **\*Active-Branch-Name)**

**git push** -u origin active-branch-name (Pushing your changes from local → remote)

It is recommended to always run the following command to get the latest

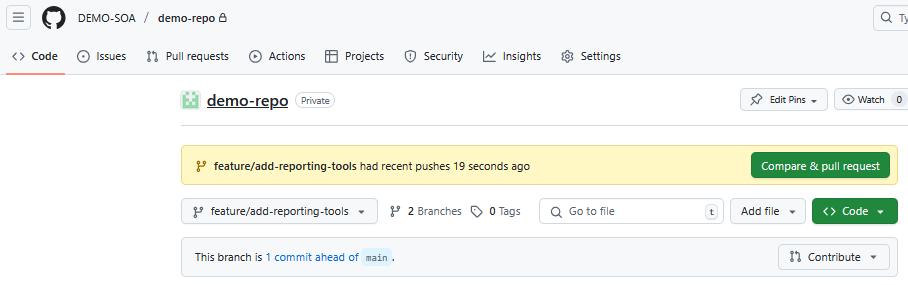
codebase into your local machine, to avoid conflicts

**git pull** (remote → local)

## **7. Create a Pull Request (PR)**

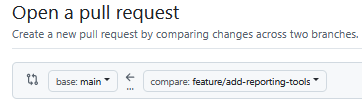
A Pull Request is a mechanism for proposing changes to a repository and initiating a review process before merging those changes into another branch (typically main).

1. After pushing your changes to your branch, navigate to your repository on GitHub.
2. GitHub will often show a banner indicating "Your branch has recent pushes. Compare & pull request." Click this button.

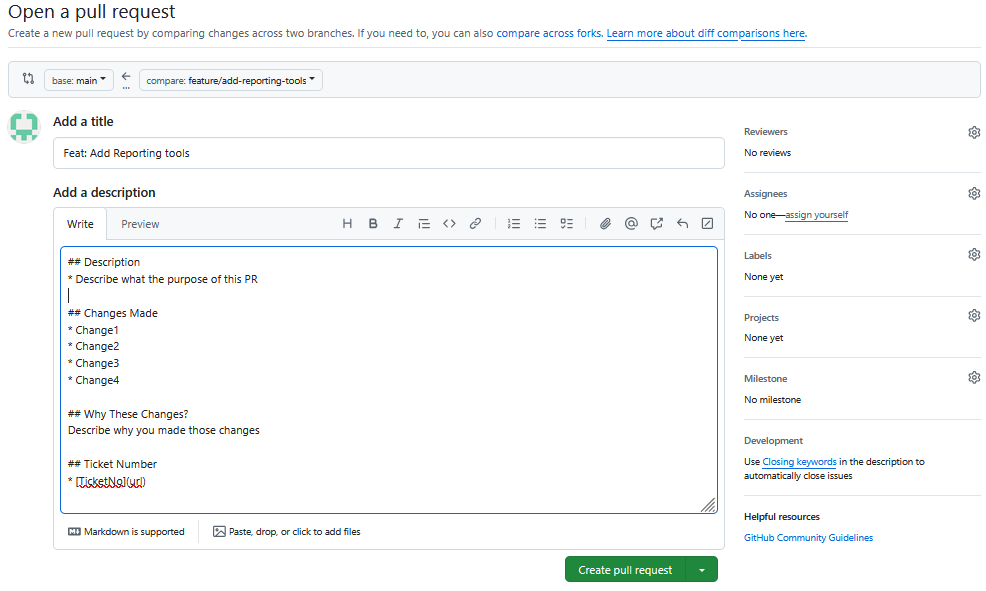


* + Alternatively, go to the **"Pull requests"** tab and click **"New pull request"**.

1. Select your branch as the "compare" branch and the target branch (e.g., main) as the "base" branch.



1. **PR Title (Sample):**
   * The title should be concise and clearly state the purpose of the PR. Follow conventional commit guidelines if applicable.
   * Example: feat: Add Reporting tools
   * Example: fix: Resolve login bug
   * Example: docs: Update README with setup instructions
2. **PR Description (Sample):**
   * The description should provide context, explain the changes, and outline any relevant tickets and details

**Structure (Best Practice):  
**

## Description

\* Describe what the purpose of this PR

## Changes Made

\* Change1

\* Change2

\* Change3

\* Change4

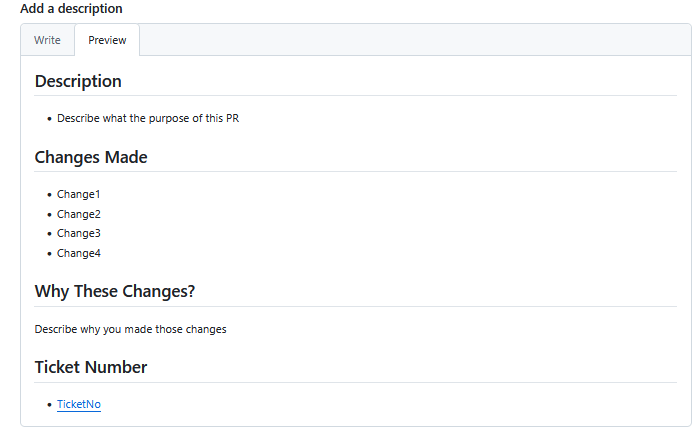
## Why These Changes?

Describe why you made those changes

## Ticket Number

\* [TicketNo](url)

You can also click on **Preview** Tab to see how the PR will look



Scroll Down to see the changes that has been made

(**-**) the negative symbol shows deleted changes and (**+**)positive shows added changes



1. **Review:**
   * Before creating, ensure you've selected the correct base and compare branches.
   * **Best Practice:** Add reviewers (team members or specific individuals) who have expertise in the affected changes. Ensure at least one approval is required by branch protection rules.
   * Add labels, assignees, and projects as needed for better organization.
2. Click **"Create pull request"**.
3. Inform your team to review and merge your Pull Request PR, below is a sample notification message  
   Hello Team,  
   Please I have submitted a Pull Request (PR) for your review and approval  
   PR Link:<https://github-url/organization-name/repo-name/pull/1>Description: PR Title and Description  
   Thank you,  
   Your Name.