

GitHub Branch Protection Setup Guide

1. Overview

[What is Branch Protection?](#)

[Why do we need it?](#)

[Problem: Human Error Risk](#)

[Solution: Branch Protection](#)

[Benefits by Role](#)

[For PM \(Anshita\):](#)

[For Developers:](#)

[For the Team:](#)

2. Prerequisites

[Required](#)

3. Workflow with Main Branch Protected

4. Setup Guide (with Screenshots)

[4.1 Access Repository Settings](#)

[4.2 Create main Branch Ruleset](#)

[Basic Settings](#)

[Rules to enable](#)

[4.3 \(Optional\) Create dev Branch Ruleset](#)

5. Validate main-protection Ruleset

[Test 1: Direct push blocked](#)

[Test 2: PR without approval blocked](#)

[Test 3: PR with approval succeeds](#)

[Test \(Edge Case\) 4: Direct edit on main branch blocked \(via GitHub Web\)](#)

1. Overview

What is Branch Protection?

i Branch Protection is a GitHub feature that prevents direct changes to critical branches (like `main` or `dev`). It enforces code review workflows by requiring pull requests and approvals before any code can be merged.

Why do we need it?

Problem: Human Error Risk

- Developers can accidentally push directly to `main` (production)
- Untested code can break the live application

- No review process = potential bugs slip through

Solution: Branch Protection

- Blocks direct pushes to protected branches
- Requires Pull Request + Approval before merge
- Ensures all code is reviewed before reaching production

Benefits by Role

For PM (Anshita):

- Visibility: All changes go through PR review before production
- Control: At least 1 reviewer's approval required before any merge
- Accountability: Clear audit trail of who approved what
- Protection: Direct push to main blocked (forces PR workflow)

For Developers:

- Clear workflow: Always create PR, never push directly
- No accidental production breaks
- Code review feedback improves code quality

For the Team:

- Better collaboration: PR comments enable discussion
- Peace of mind: Production is protected from human error
- Consistent workflow: System automatically guides everyone through the same PR process
- Extensible workflow: Can integrate automated tests, linting, or CI checks into the PR process

2. Prerequisites

Required

- GitHub Pro (\$4/month) or GitHub Team plan
- Free plan does not support Branch Protection on private repositories

3. Workflow with Main Branch Protected

- Before: Anyone can push to main
- After: PR + Approval required



⚠ Note: This document covers **main branch protection only** (Production Phase).

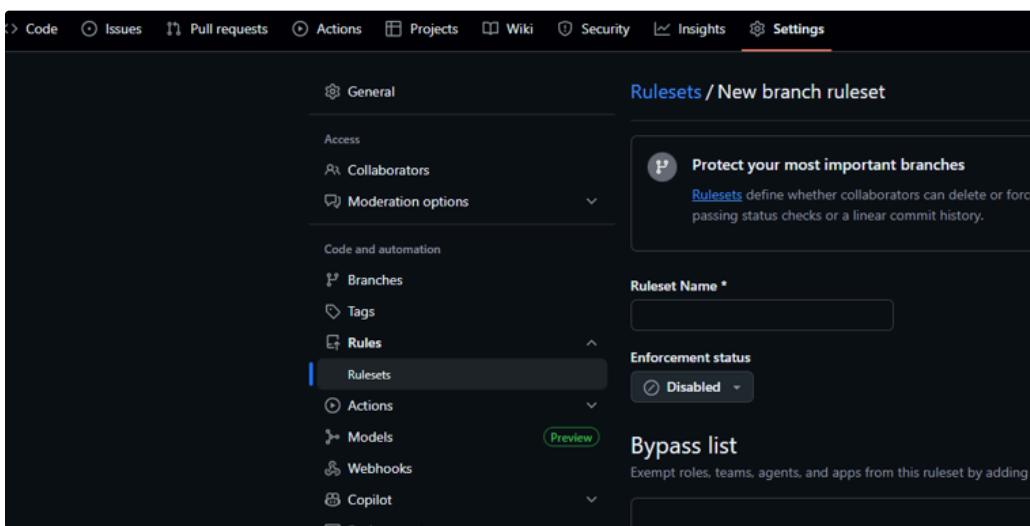
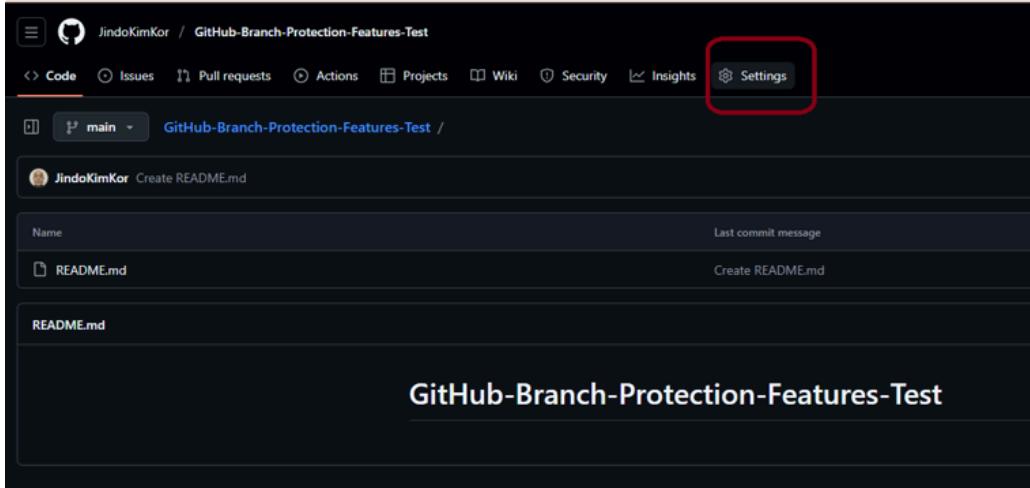
Protecting dev branch is optional and can be configured later based on team needs.

Adding approval requirements to dev may slow down the development flow, so it's recommended to start with main protection first and evaluate dev protection after observing the workflow.

4. Setup Guide (with Screenshots)

4.1 Access Repository Settings

1. Go to your repository on GitHub
2. Click Settings tab
3. Navigate to Rules → Rulesets
4. Click New ruleset → New branch ruleset



4.2 Create main Branch Ruleset

Basic Settings

- Ruleset Name: main-protection
- Bypass list: Repository admin → Anshita
- Enforcement Status: Active
- Target Branches: Add Target → Include by pattern → main

Rules to enable

- Restrict deletion:
 - Prevent branch from being deleted

- Require a pull request before merging
 - No direct push allowed to main, even if anyone make a change on GitHub directly, except Anshita
 - Required approvals: 1
 - At least 1 reviewers must approve
 - → Developers can not merge without an approval
 - Dismiss stale approvals
 - → If changes are made to an approved PR, the approval is dismissed. This ensures no unreviewed code gets merged.
- Block force pushes
 - Prevent history rewriting

The screenshot shows the GitHub Ruleset configuration interface. The ruleset is named "main-protection" and is set to "Active". In the "Bypass list" section, the "Repository admin" role is selected. In the "Target branches" section, the "main" branch is listed under "Branch targeting criteria". On the right, a modal window titled "Add bypass" is open, showing options to choose roles, teams, agents, and apps that can bypass the ruleset. The "Repository admin" role is checked and highlighted with a blue border. Other options like "Deploy keys" and "Maintain" role are also listed.

The screenshot shows the GitHub Target branches configuration interface. It lists the "main" branch under "Branch targeting criteria". In the "Rules" section, there is a note: "Which rules should be applied?". On the right, a sidebar titled "Target by inclusion or exclusion pattern" offers options to "Include default branch", "Include all branches", "Include by pattern" (which is selected and highlighted with a blue border), and "Exclude by pattern".

Branch rules

Restrict creations

Only allow users with bypass permission to create matching refs.

Restrict updates

Only allow users with bypass permission to update matching refs.

Restrict deletions

Only allow users with bypass permissions to delete matching refs.

Require linear history

Prevent merge commits from being pushed to matching refs.

Require a pull request before merging

Require all commits be made to a non-target branch and submitted via a pull request before they can be merged.

Hide additional settings ^

Required approvals

1 ▾

The number of approving reviews that are required before a pull request can be merged.

Dismiss stale pull request approvals when new commits are pushed

New, reviewable commits pushed will dismiss previous pull request review approvals.

Require review from specific teams Preview

A collection of reviewers and associated file patterns. Each reviewer has a list of file patterns which determine the files that reviewer is required to review.

Require review from Code Owners

Require an approving review in pull requests that modify files that have a designated code owner.

Require approval of the most recent reviewable push

Whether the most recent reviewable push must be approved by someone other than the person who pushed it.

Require conversation resolution before merging

All conversations on code must be resolved before a pull request can be merged.

Allowed merge methods

Merge, Squash, Rebase ▾

When merging pull requests, you can allow any combination of merge commits, squashing, or rebasing. At least one option must be enabled.

The screenshot shows the 'checks' section of a GitHub repository settings page. It lists several protection rules:

- Block force pushes**
Prevent users with push access from force pushing to refs.
- Require code scanning results**
Choose which tools must provide code scanning results before the reference is updated. When configured, code scanning must be enabled and have results for both the commit and the reference being updated.
- Require code quality results**
Choose which severity levels of code quality results should block pull request merges. When configured, a code quality analysis must be done on the pull request before the changes can be merged.
- Automatically request Copilot code review**
Request Copilot code review for new pull requests automatically if the author has access to Copilot code review and their premium requests quota has not reached the limit.
- Manage static analysis tools in Copilot code review** (Preview)
Copilot code review will include findings from the selected static analysis tools in its review comments.

A green 'Create' button is located at the bottom left of the form.

4.3 (Optional) Create dev Branch Ruleset

- 💡** Protecting dev branch is optional. Since dev environment mistakes don't affect production, you may want to keep dev branch unprotected for faster iteration. Decide based on team workflow needs.

If you choose to protect dev branch, you have options:

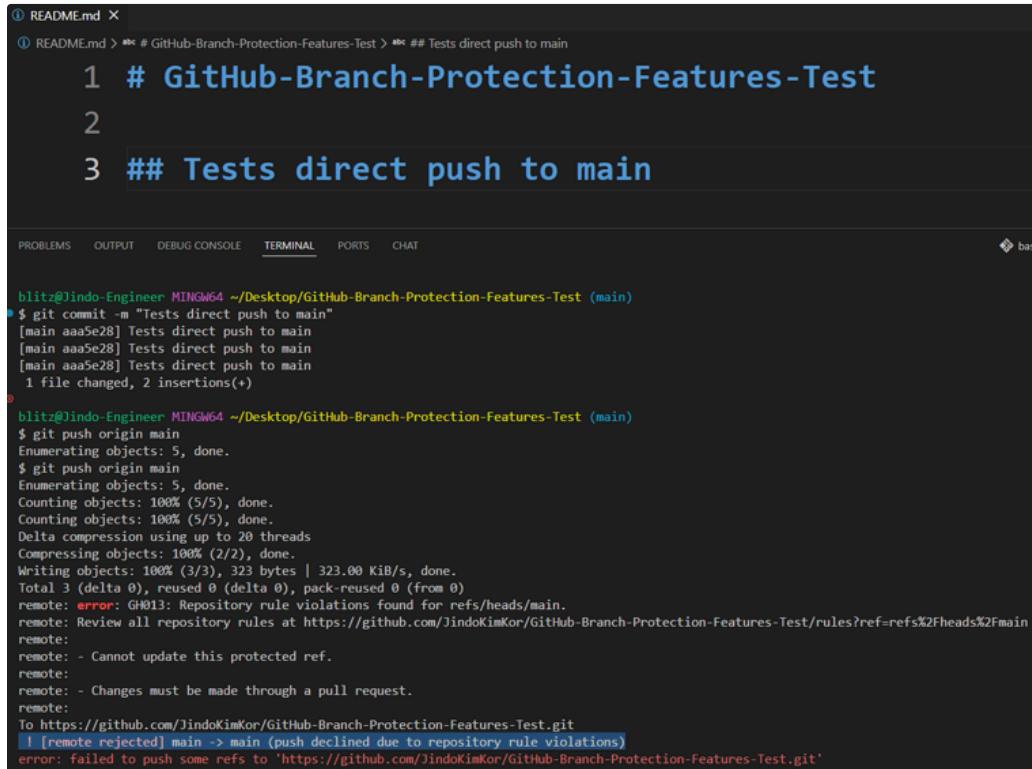
- Option A: Same as main (strict)
 - Same rules as main branch
 - All PRs require approval
- Option B: PR required, no approval
 - Ruleset name: `dev-protection`
 - Target branches: `dev`
 - Rules:
 - Require a pull request before merging
 - Required approvals: 0
 - Block force pushes

- ℹ️** This could ensure that code changes are tracked via PRs without slowing down development.

5. Validate main-protection Ruleset

After creating the ruleset, verify it works correctly:

Test 1: Direct push blocked



```
① README.md ×
① README.md > ✘ # GitHub-Branch-Protection-Features-Test > ✘ ## Tests direct push to main

1 # GitHub-Branch-Protection-Features-Test
2
3 ## Tests direct push to main

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CHAT
bash

blitz@Jindo-Engineer MINGW64 ~/Desktop/GitHub-Branch-Protection-Features-Test (main)
$ git commit -m "Tests direct push to main"
[main aaa5e28] Tests direct push to main
[main aaa5e28] Tests direct push to main
[main aaa5e28] Tests direct push to main
1 file changed, 2 insertions(+)

blitz@Jindo-Engineer MINGW64 ~/Desktop/GitHub-Branch-Protection-Features-Test (main)
$ git push origin main
Enumerating objects: 5, done.
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Counting objects: 100% (5/5), done.
Delta compression using up to 20 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 323 bytes | 323.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote: error: GH013: Repository rule violations found for refs/heads/main.
remote: Review all repository rules at https://github.com/JindoKimKor/GitHub-Branch-Protection-Features-Test/rules?ref=refs%2Fheads%2Fmain
remote:
remote: - Cannot update this protected ref.
remote:
remote: - Changes must be made through a pull request.
remote:
To https://github.com/JindoKimKor/GitHub-Branch-Protection-Features-Test.git
 ! [remote rejected] main -> main (push declined due to repository rule violations)
error: failed to push some refs to 'https://github.com/JindoKimKor/GitHub-Branch-Protection-Features-Test.git'
```

1. Make changes locally and try to push directly to main

2. Expected: Push rejected with error message

It works as expected

Test 2: PR without approval blocked

The screenshot shows a terminal window with the following content:

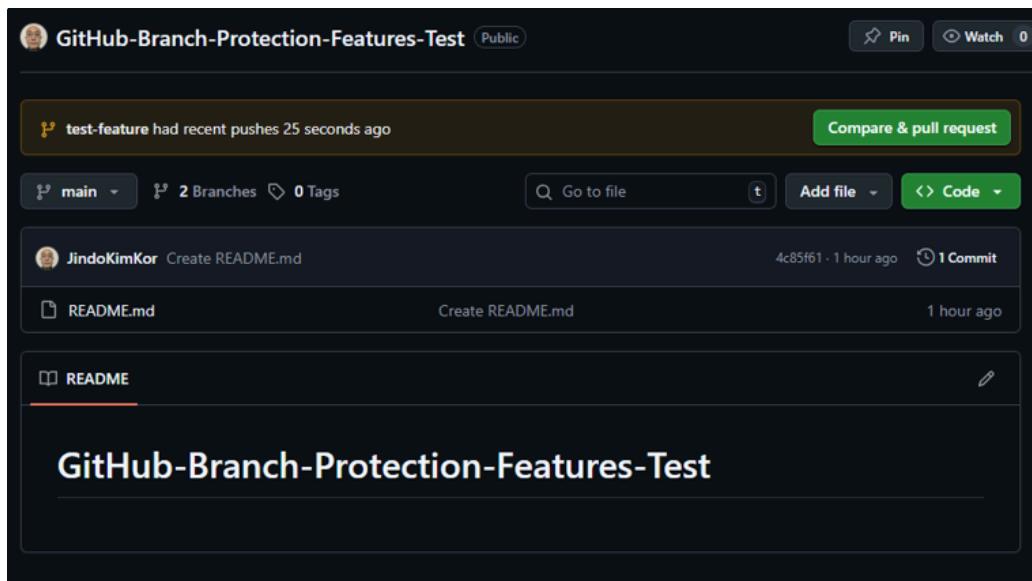
```
① README.md ×
① README.md > abc # GitHub-Branch-Protection-Features-Test > abc ## Tests push to test-feature branch
  1 # GitHub-Branch-Protection-Features-Test
  2
  3 ## Tests push to test-feature branch

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS CHAT
To push the current branch and set the remote as upstream, use

git push --set-upstream origin test-feature
To have this happen automatically for branches without a tracking
upstream, see 'push.autoSetupRemote' in 'git help config'.

blitz@Jindo-Engineer MINGW64 ~/Desktop/GitHub-Branch-Protection-Features-Test (test-feature)
$ git push origin test-feature
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 20 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 318 bytes | 318.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/JindoKimKor/GitHub-Branch-Protection-Features-Test.git
  aaa5e28..c474eae test-feature -> test-feature

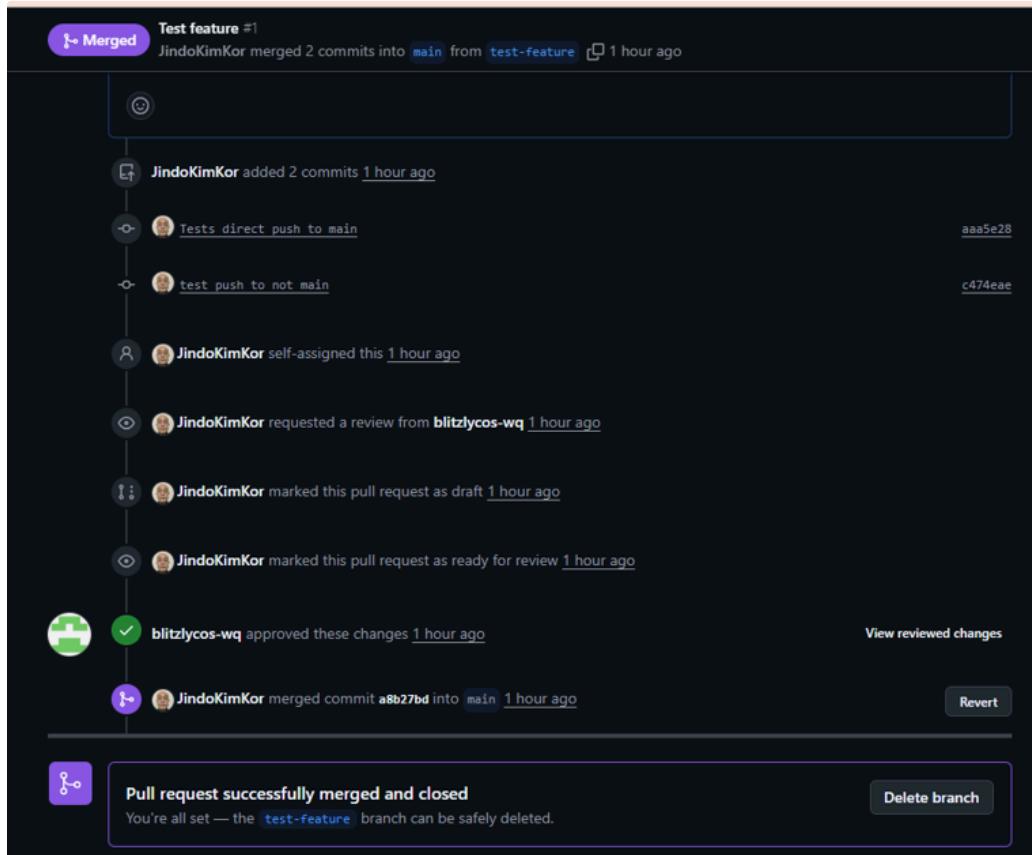
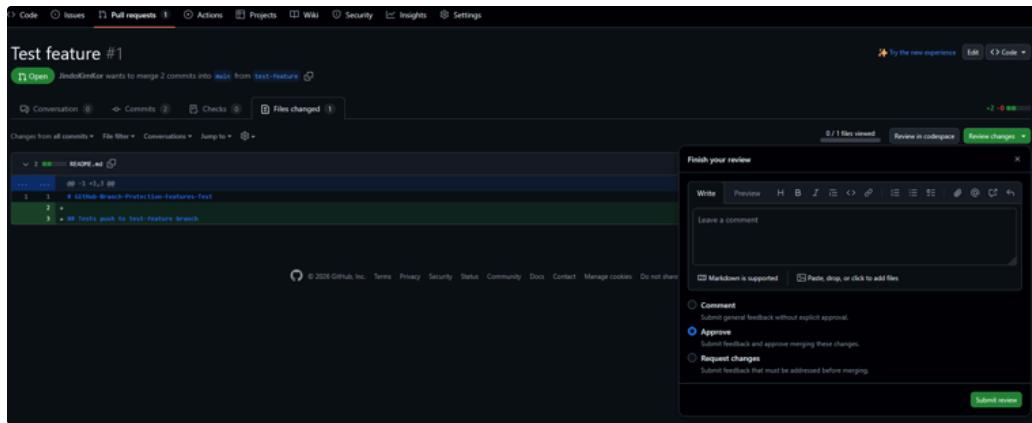
blitz@Jindo-Engineer MINGW64 ~/Desktop/GitHub-Branch-Protection-Features-Test (test-feature)
$
```



1. Create a PR to main
2. Try to merge without approval
3. Expected: Merge button is disabled with message "Review required"

It works as expected

Test 3: PR with approval succeeds

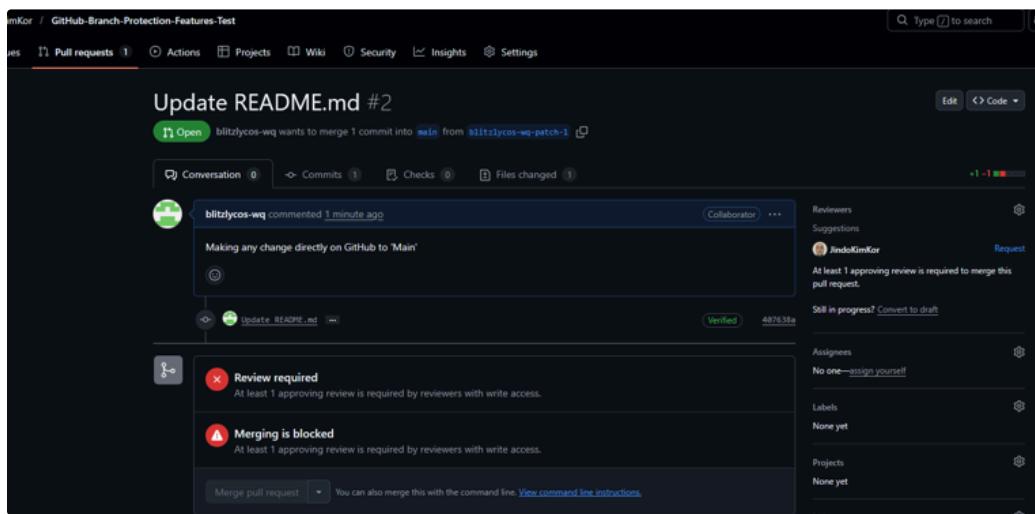
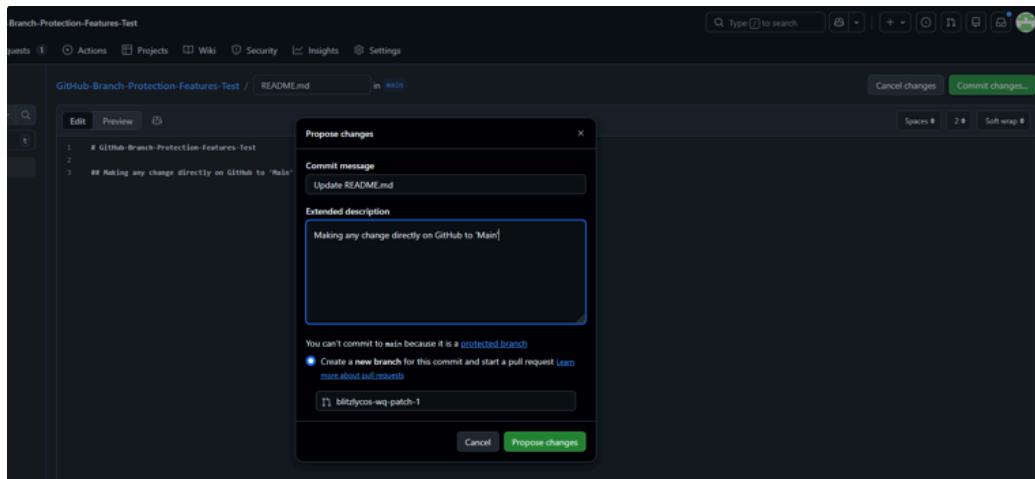


1. Have another team member approve the PR

2. Expected: Merge button becomes enabled

It works as expected

Test (Edge Case) 4: Direct edit on main branch blocked (via GitHub Web)



1. Try to edit a file directly on main branch via GitHub web
2. Expected: GitHub creates a new branch and prompts you to create a PR

It works as expected