

GitHub Pull Request and Review Governance Guide

Contents

- Understanding the Pull Request Lifecycle..... 3
- Branching Governance and Conventions..... 3
- Creating a Pull Request..... 3
- Reviewing a Pull Request..... 3
- Resolving Merge Conflicts..... 4
- Pull Request Merge Strategies..... 4
- Pull Request Status Indicators..... 4

Understanding the Pull Request Lifecycle

A pull request proposes changes from a working branch to the primary repository branch, enabling peer review and automated testing before integration.

In GitHub, pull requests serve as the primary mechanism for code and documentation review.

A pull request progresses through three primary stages:

- **Draft State:** Work in progress; automated checks run, but peer review is not required.
- **Open for Review:** Ready for peer review and automated continuous integration status checks within the target 24 hours SLA window.
- **Approved & Merged:** Required reviewers approve changes, all automated status checks pass, and code merges into the target branch.

Branching Governance and Conventions

Branching strategies establish consistent naming patterns and isolate active development from production-ready code.

In GitHub, long-lived default branches are protected from direct commits to preserve release stability.

Contributors must isolate changes in short-lived feature or bugfix branches adhering to standardized naming prefixes:

- **feature/:** Work supporting new capabilities, APIs, or major documentation updates.
- **fix/:** Targeted repairs for reported defects or incorrect guidance.
- **docs/:** Minor prose improvements, typo corrections, or maintenance.

Creating a Pull Request

Initiate a formal review request to propose branch changes into the primary repository.

Ensure all local commits are pushed to your topic branch in acme-docs/knowledge-base.

Important: Direct commits to the default branch are blocked by repository protection rules. All changes must originate from a pull request.

1. Navigate to the repository homepage in GitHub.
2. Click **Compare & pull request** alongside your recently pushed branch.
3. Provide a descriptive title and fill out the pull request body template.
4. Assign relevant code owners and reviewers, then click **Create pull request**.

Reviewing a Pull Request

Evaluate proposed changes, leave line-by-line feedback, and submit an approval or change request.

You must have read access to acme-docs/knowledge-base.



Warning: Force-pushing to a shared branch overwrites commit history and can break active code reviews for other team members.

1. Navigate to the **Pull Requests** tab in GitHub.
2. Select the pull request you want to review and open the **Files changed** tab.

3. Hover over a specific line of code or Markdown and click the plus icon to add an inline comment.
4. Verify that required status checks and code owner approvals meet internal repository policy prior to merging.
5. Click **Review changes** and select one of the following submission types:
 - **Comment:** General feedback without explicit approval or rejection.
 - **Approve:** Confirms the changes are ready to merge.
 - **Request changes:** Requires feedback resolution before merging (Enforced for repository maintainers).

Resolving Merge Conflicts

Reconcile competing branch changes when target branch updates collide with proposed edits.

Merge conflicts occur when overlapping lines are modified independently across branches.

1. Fetch the latest changes from the default branch into your working environment.
2. Rebase or merge the target branch into your feature branch locally.
3. Open conflicting files, locate the conflict markers (<<<<<<), and select the correct code or text block.
4. Stage the resolved files and complete the merge commit.
5. Push the updated branch history to update the active pull request automatically.

Pull Request Merge Strategies

A reference of supported branch merge strategies, commit preservation rules, and recommended use cases for repository maintainers.

Supported Merge Methods

Select the appropriate merge method based on branch lifetime and commit history requirements:

Strategy	Commit History Impact	Recommended Use Case
Create a Merge Commit	Preserves all individual branch commits and appends an explicit merge commit.	Feature branches requiring complete historical traceability during audits.
Squash and Merge	Combines all branch commits into a single unified commit on the target branch.	Documentation edits, minor bug fixes, or long-running PRs with noisy histories.
Rebase and Merge	Reapplies individual commits onto the target branch sequentially without a merge commit.	Linear repository histories where clean individual commit tracking is required.

Pull Request Status Indicators

Reference table of continuous integration status indicators and merge readiness states.

Status Badges

State	Description	Required Action
Draft	PR is open for active editing and testing.	Mark as ready for review when complete.
Checks Pending	Automated continuous integration linters are running.	Wait for build pipeline completion.

State	Description	Required Action
Changes Requested	A designated reviewer requested updates.	Address feedback and re-request review.