

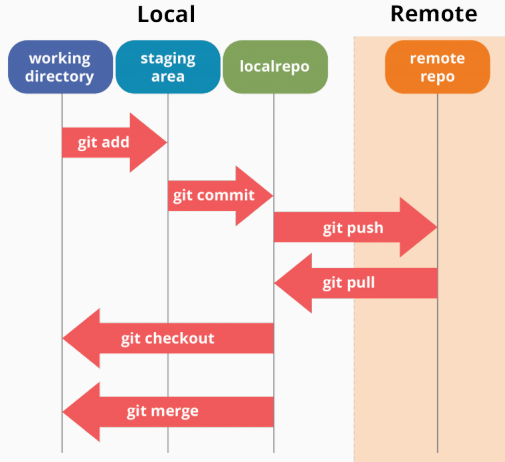
git/GitHub for Developers

February 15, 2024

Engineers for Exploration, UC San Diego

Introduction

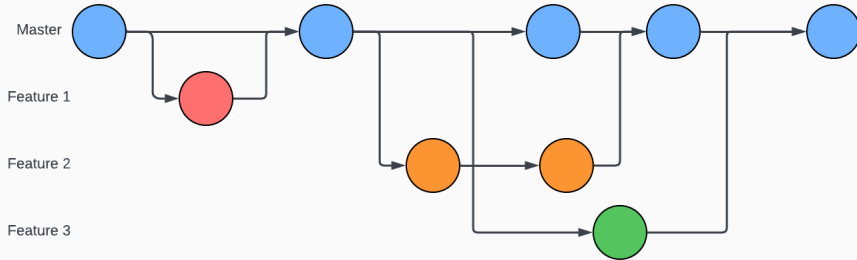
- **Git vs. GitHub:** Distributed VCS vs. collaboration platform.
- **Purpose:** Enhances project management and teamwork.



- Feature Branch Workflow
- Gitflow Workflow
- Fork Workflow

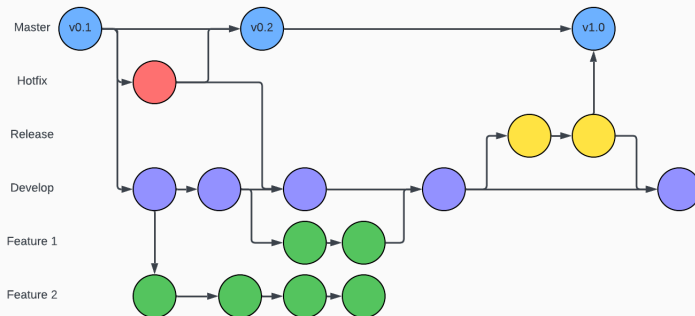
Feature Branch Workflow

- Develop each feature in its own branch.
- Merges via pull requests for code review.
- Keeps main branch stable, encourages collaboration.
- Ideal for projects with simultaneous feature development.



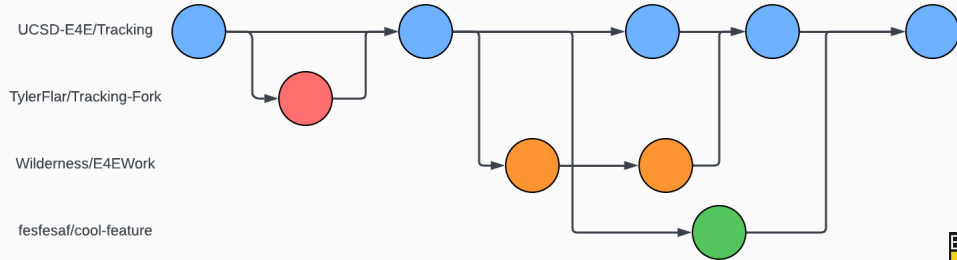
Gitflow Workflow

- Structured model: development, features, releases, hotfixes.
- Systematic release management, clear branch roles.
- Tracks progress efficiently, supports parallel releases.
- Suited for scheduled release cycles.



Fork Workflow

- Developers work on personal repository copies.
- Changes proposed via pull requests.
- Encourages external contributions, safe experimentation.
- Ideal for open-source and large collaborations.



- Marks significant project milestones.
- Useful for release points, use semantic versioning.
- Lightweight tags: `git tag tagname`.
- Annotated tags: `git tag -a tagname "message"`.
- List/delete tags: `git tag`, `git tag -d tagname`.

Pull Requests (PR) Management

- Keep PRs small for easy review.
- Use checklists for consistent reviews.
- Automate tests and checks via GitHub Actions.

- Draft new release, choose git tag.
- Add release notes describing changes.
- Bundles code, executables, and assets.
- Detailed notes inform users of updates.
- Example: <https://github.com/HumanSignal/label-studio/releases>

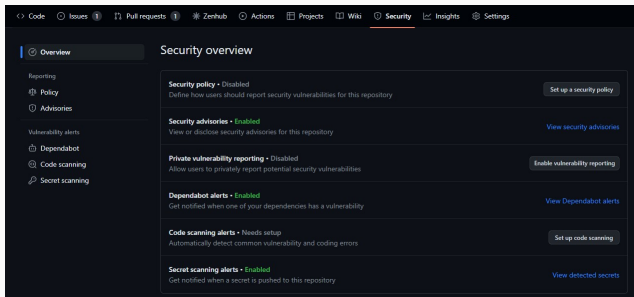
- Triggered by GitHub events (push, PRs).
- Workflows combine actions in YAML files.
- Runs on GitHub-hosted or self-hosted runners.
- Automates tests and deployment on PR merge.
- Auto-assigns issues, auto-labels PRs by path.

Putting it Together

- Combine Tags, Releases, and GitHub Actions.
- Interact with source code.
- Example: <https://github.com/TylerFlar/MinecraftDiscord-CrossChat>

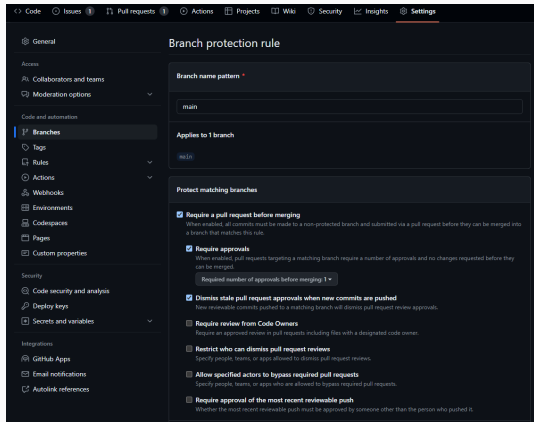
Protection against:

- Vulnerable dependencies
- Some code vulnerabilities
- Some committed secrets



Why do we need branch protections?

Configuring Branch Protection



The screenshot shows the GitHub 'Branch protection rule' configuration page. The left sidebar contains navigation links for General, Access, Collaborators and teams, Moderation options, Code and automation, Branches (selected), Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages, Custom properties, Security, Code security and analysis, Deploy keys, and Secrets and variables. The main content area is titled 'Branch protection rule' and includes sections for 'Branch name pattern' (set to 'main'), 'Applies to 1 branch' (set to 'main'), and 'Protect matching branches'. Under 'Protect matching branches', several checkboxes are enabled: 'Require a pull request before merging', 'Require approvals' (with a dropdown set to 'Required number of approvals before merging: 1'), 'Dismiss stale pull request approvals when new commits are pushed', 'Require review from Code Owners', 'Restrict who can dismiss pull request reviews', 'Allow specified actors to bypass required pull requests', and 'Require approval of the most recent reviewable push'.

☒ **Require status checks to pass before merging**
Choose which [status checks](#) must pass before branches can be merged into a branch that matches this rule. When enabled, commits must first be pushed to another branch, then merged or pushed directly to a branch that matches this rule after status checks have passed.

☐ **Require branches to be up to date before merging**

This ensures pull requests targeting a matching branch have been tested with the latest code. This setting will not take effect unless at least one status check is enabled (see below).

Search for status checks in the last week for this repository

Status checks that are required.

☒ **Require conversation resolution before merging**

When enabled, all conversations on code must be resolved before a pull request can be merged into a branch that matches this rule. [Learn more about requiring conversation completion before merging.](#)

☐ **Require signed commits**

Commits pushed to matching branches must have verified signatures.

☐ **Require linear history**

Prevent merge commits from being pushed to matching branches.

☐ **Require merge queue**

Merges to matching branches must be performed via a merge queue.

☐ **Require deployments to succeed before merging**

Choose which environments must be successfully deployed to before branches can be merged into a branch that matches this rule.

☐ **Lock branch**

Branch is read-only. Users cannot push to the branch.

☐ **Do not allow bypassing the above settings**

The above settings will apply to administrators and custom roles with the "bypass branch protections" permission.

☐ **Restrict who can push to matching branches**

Specify people, teams, or apps allowed to push to matching branches. Required status checks will still prevent these people, teams, and apps from merging if the checks fail.

Branch Protection Example

`https://github.com/UCSD-E4E/branch_protections_demo`

- Interactive Rebase*: Rewrite history, edit commits.
- Stashing*: Temporarily shelf changes for tasks.
- Cherry-picking: Apply commit to another branch.
- * Not recommended for shared/remote repositories.

Cherry-picking

- *git cherry-pick commit-hash*

