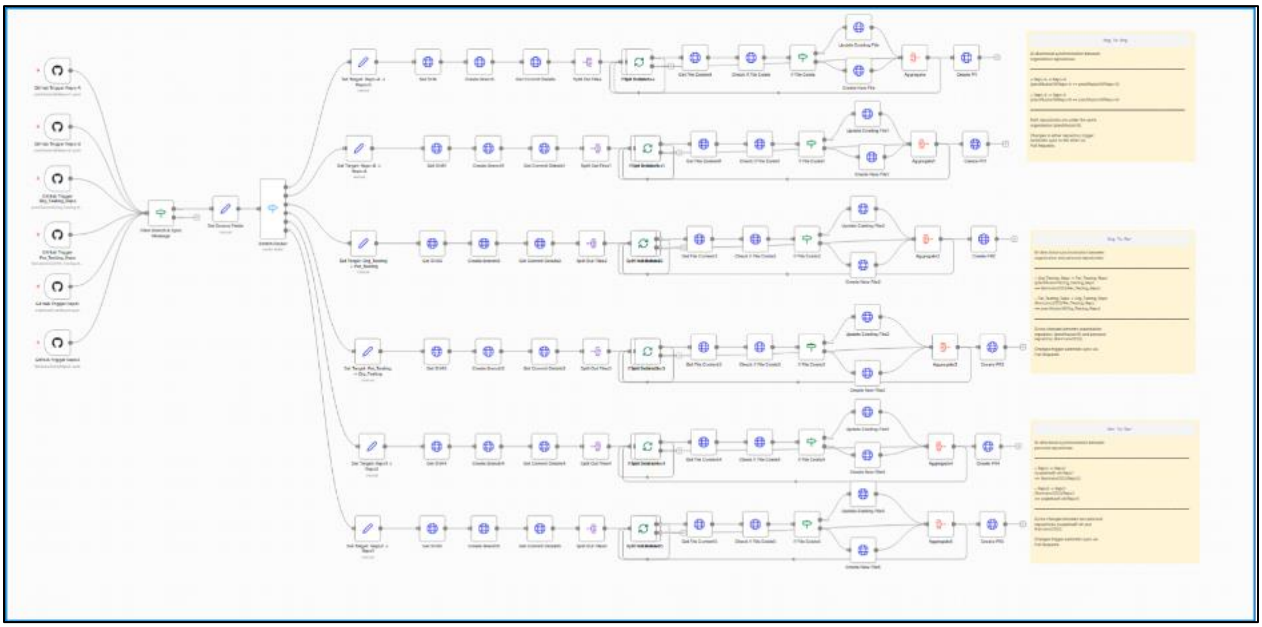
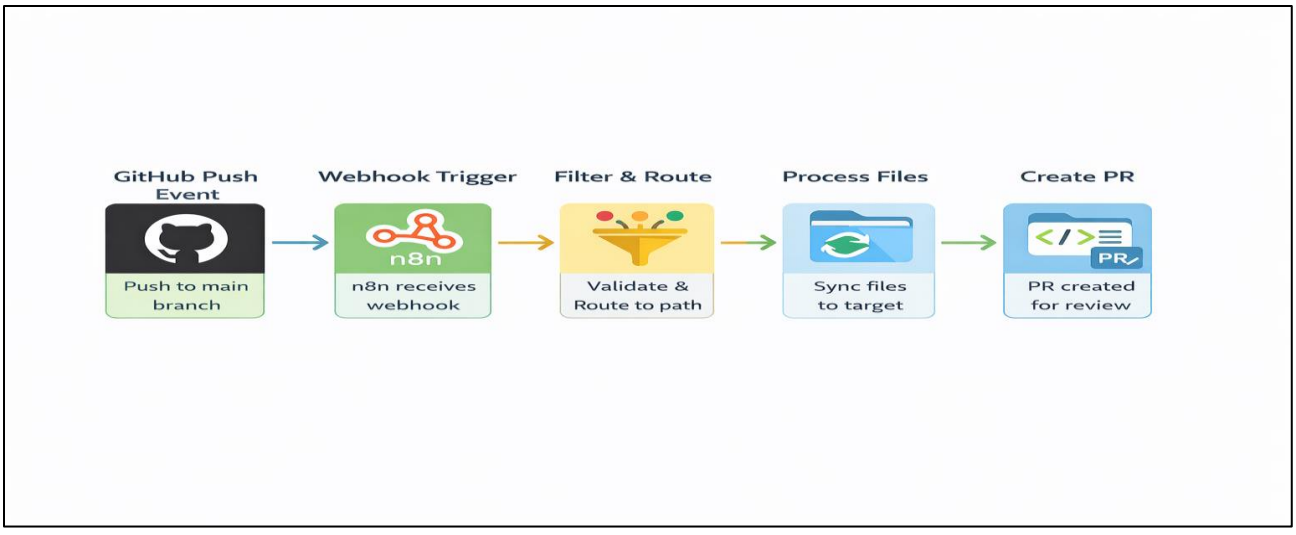
Bi-Directional GitHub Repository Sync

Workflow



Workflow Architecture



Executive Summary:

This solution delivers an end-to-end, automated mechanism to keep multiple GitHub repositories continuously synchronized, without requiring teams to change how they workday to day. Developers keep using their normal git push workflow, and the system takes care of the rest: it detects changes, mirrors the relevant files into the paired repository, and opens a clearly labeled Pull Request for review. This turns repository synchronization from a manual, error-prone chore into a predictable, standardized process. By centralizing sync logic in a single workflow, organizations gain consistent behavior across different repository pairs—whether they are org-to-org, org-to-personal, or personal-to-personal. Every sync is transparent and auditable through GitHub PRs and commit history, so leads and managers can see exactly what changed, when, and why. The result is faster collaboration across teams, less risk of code drift, and a repeatable governance model that scales as more repositories and teams are added.

Business Problem Statement:

Modern engineering teams often depend on multiple repositories that must stay aligned—shared libraries, service contracts, common configuration, or client-specific forks. Without automation, keeping these repos in sync relies on developers manually copying changes, re-implementing fixes, or re-creating PRs in multiple places. This manual effort is slow, unstructured, and easy to forget, which leads to inconsistent behavior between environments, duplicated bugs, and surprises during deployments. Over time, this lack of structured synchronization creates real business risk. Critical patches may be applied in one repo but never reach another, compliance fixes may be missed in downstream clients, and teams waste hours chasing "why does it work here but not there?" issues. Leadership has little visibility into what was synced, when it happened, and whether all dependent repositories are actually up to date. The result is increased operational overhead, higher defect rates, and slower delivery of features to customers.

Solution Overview:

The Bi-Directional GitHub Repo Sync Workflow addresses these challenges by acting as an always-on automation layer between repositories. It listens to push events on the main branch of each repo and, whenever a real developer change is detected, it determines which paired repository should receive that update. It then prepares a dedicated sync branch in the target repo, copies over only the files changed in that commit, and opens a Pull Request summarizing the change, including the original commit message, SHA, and author. From the user's perspective, no new tools or commands are required: they simply commit and push as usual, and the system ensures that relevant changes appear as PRs in the paired repositories. Built-in safeguards prevent infinite loops by recognizing and ignoring its own sync-generated commits, ensuring that only genuine human-initiated changes trigger new syncs. Because everything flows through GitHub PRs, teams retain full control over when and how changes are merged.

Scope:

The Bi-Directional GitHub Repository Sync Workflow automates two-way synchronization between GitHub repositories using n8n. It supports six bi-directional sync paths across three categories: Organization to Organization (Repo-A ↔ Repo-B), Organization to Personal (Org_Testing_Repo ↔ Per_Testing_Repo), and Personal to Personal (Repo1 ↔ Repo2). The workflow monitors push events on the main branch, processes multiple files per commit, and creates Pull Requests in target repositories for review. It handles file creation and updates, skips deleted files, and prevents sync loops by filtering commits with "Sync from Repo" in the message. The workflow processes files sequentially to respect GitHub API rate limits and preserves commit metadata including author information and commit SHA. It does not auto-merge PRs, resolve conflicts, or handle non-main branches. The solution is designed for small to medium-sized repositories requiring automated synchronization with manual review and approval processes.

Key Business Benefits:

- **No more manual syncing:** Changes automatically flow between paired repos
- **Fewer merge conflicts:** Repos stay current instead of drifting apart over time
- **Standard review process:** All syncs go through Pull Requests, not silent background pushes
- **Developers stay focused:** No time wasted managing sync branches manually
- **Loop-safe:** Smart filters prevent the system from syncing its own syncs
- **Works everywhere:** Supports org-to-org, org-to-personal, and personal-to-personal repo pairs
- **Clear audit trail:** Every sync is visible in GitHub's PR and commit history

Use Cases / Scenarios:

- Keep shared libraries synchronized between frontend and backend repos
- Mirror updates between an internal template and multiple client forks
- Sync a private internal repo with its public open-source version
- Enable developers to work in personal repos while keeping the org repo current
- Gradually split a monorepo into services while keeping shared code in sync

Target Audience:

- Engineering teams maintaining multiple related codebases
- DevOps/Platform teams automating cross-repo workflows
- CTOs and Tech Leads standardizing how code moves between repositories
- SaaS companies with multi-repo or multi-tenant architectures

Security & Compliance Overview:

The workflow uses official GitHub APIs with Personal Access Tokens stored securely in n8n's credential vault—never in plain text. All changes go through Pull Requests, ensuring every modification is reviewed before merging. Nothing happens silently. Every sync action is fully auditable through GitHub's standard PR timeline and commit logs, supporting compliance and change-management requirements

Competitive Advantage / Differentiation:

- **True bi-directional sync:** Both directions work for each repo pair, not just one-way mirroring
- **Built-in loop prevention:** Won't trigger infinite sync cycles
- **PR-based governance:** Fits into existing approval workflows instead of bypassing them
- **No custom GitHub apps:** Works with standard tokens and webhooks—simpler setup, less maintenance
- **Flexible pairing:** Any repos can be paired or re-paired with quick configuration changes
- **Multi-file aware:** Handles commits with many files and multiple repo pairs in one workflow

Implementation & Onboarding (Client-Friendly):

Most teams deploy this in 1–2 hours with GitHub access and n8n already set up. Import the workflow, update repo names and routing rules to match your repositories, configure credentials, and enable GitHub webhooks. The same workflow can be cloned for additional repo pairs—just adjust a few nodes. A simple test (push one file change, verify the branch and PR) validates everything before production use.

Positioning Guidance (Optional):

- Marketing/Proposal: Lead with Executive Summary, Business Problem, Key Benefits, and Use Cases
- Client Handover: Emphasize Solution Overview, Security & Compliance, and Implementation steps
- Sales Enablement: Focus on Business Problem, Competitive Advantage, and Benefits

Credentials to Add:

1. GitHub Personal Access Tokens

This workflow requires **GitHub Personal Access Tokens (PATs)** with appropriate repository access. The following credentials must be created and configured in **n8n**.

1.1 GitHub Account 1:

Used for: GitHub webhook triggers for Repo-A, Repo-B, Org_Testing_Repo, Per_Testing_Repo, and Repo2

Account: GitHub account with access to the **ptechfusion19** organization

Required Permissions: repo – Full control of private repositories

Repositories Access Required:

- ptechfusion19/Repo-A
- ptechfusion19/Repo-B
- ptechfusion19/Org_Testing_Repo
- Ramzanx0553/Per_Testing_Repo
- Ramzanx0553/Repo2

1.2 GitHub Account 2

Used for: GitHub webhook trigger for **Repo1**

Account: wajeehaafi-alt

Required Permissions: repo – Full control of private repositories

Repositories Access Required: wajeehaafi-alt/Repo1

1.3 Header Authentication – GitHub (Account 1)

Used for: Most GitHub API HTTP requests, including:

- Get SHA
- Create branch
- Get commit details
- Get file content
- Check if file exists
- Update existing file

Authentication Type:

- HTTP Header Authentication
- Bearer Token

Header Format: Authorization: Bearer [TOKEN]

Required Permissions:

- repo – Full control of private repositories
- read:org – Read organization and team membership (required for org repositories)

Repositories Access Required:

- ptechfusion19/Repo-A
- ptechfusion19/Repo-B
- ptechfusion19/Org_Testing_Repo
- Ramzanx0553/Per_Testing_Repo
- Ramzanx0553/Repo2
- wajeehaafi-alt/Repo1

1.4 Header Authentication – GitHub (Account 2)

Used for: Specific operations in the **Repo2 → Repo1** synchronization flow, including:

- Get SHA
- Create branch
- Create new file
- Create pull request

Authentication Type: HTTP Header Authentication and Bearer Token

Header Format: Authorization: Bearer [TOKEN]

Required Permissions: repo – Full control of private repositories

Repositories Access Required: wajeehaafi-alt/Repo1

2. How to Generate a GitHub Personal Access Token

1. Go to **GitHub → Settings → Developer settings → Personal access tokens → Tokens (classic)**
2. Click **"Generate new token (classic)"**
3. Set **Token Name:** n8n GitHub Sync Workflow – [Account Name]
4. Set **Expiration:** Choose an appropriate duration
5. Select the following scopes:
    a. repo (Full control of private repositories)
    b. repo:status
    c. repo_deployment
    d. public_repo
    e. repo:invite
    f. security_events
    g. admin:repo_hook (Full control of repository hooks)
    h. workflow (Update GitHub Actions workflows) – *Optional*
6. Click **"Generate token"**
7. **Copy and store the token securely** (it will not be shown again)

Conclusion

The Bi-Directional GitHub Repository Sync Workflow removes the manual overhead of keeping multiple repositories synchronized. By automating file synchronization through Pull Requests, it preserves your review process while eliminating copy-paste errors, reducing code drift, and providing a clear audit trail. Whether you're managing organization-to-organization repositories, coordinating between personal and organizational repos, or maintaining consistency across multiple client repositories, this solution scales with your needs. With few setups and built-in safeguards against sync loops, it delivers immediate productivity gains while maintaining the governance and security standards your team requires.