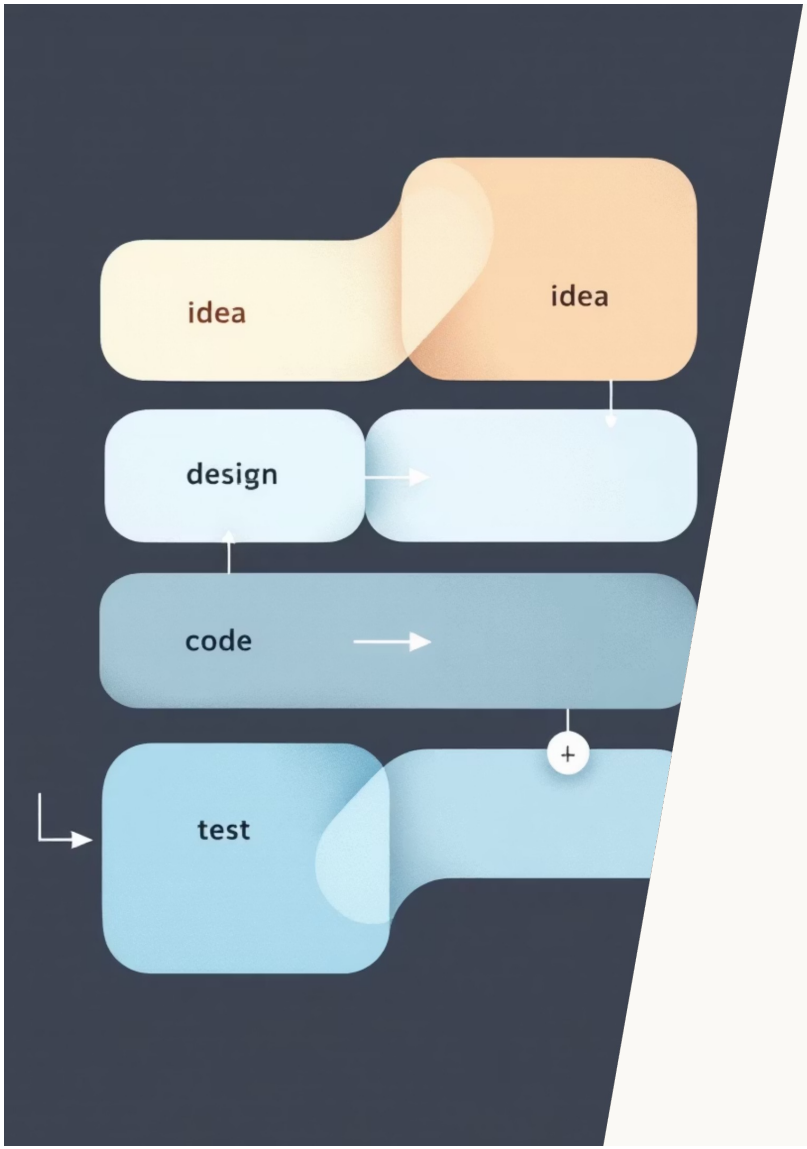
GitHub & Team Collaboration

Master GitHub workflows to build safer, faster, and more efficient engineering teams through proven collaboration practices.



Course Overview

Transforming Team Development

GitHub extends beyond version control—it's a comprehensive collaboration platform that enables teams to work in parallel, maintain code quality, automate deployments, and ensure production stability. This course explores strategic workflows that transform how engineering teams ship software.

Branching Strategies for Team Success

Feature Branches

Isolate each task for parallel development without conflicts. Every feature gets its own safe workspace.

Dev Branch

Integration and staging area where features come together for testing before production release.

Main Branch

Production-ready code only. Protected, stable, and always deployable with confidence.

This three-tier workflow—Feature → Dev → Main—enables parallel work, easy rollbacks, and clear review paths whilst maintaining production integrity.

Learn more: Git Branching | Feature Branches | Git Workflow

Protecting Your Production Code

Branch Protection Rules

Safeguard your main branch with automated enforcement:

- Require peer approvals before merging
- Mandate passing CI/CD tests
- Restrict direct pushes to production
- Enforce signed commits for traceability

Learn more: Branch Protection | GitHub Protection Rules | Securing Main Branch

These protection rules create a safety net that prevents untested code from reaching production whilst maintaining development velocity.

Pull Requests & Peer Reviews

01

Create PR with Template

Include description, purpose, linked tickets, and testing information for complete context.

02

Automated Checks Run

CI pipeline validates linting, tests, security scans, and build status automatically.

03

Peer Review

Team members verify logic correctness, coding standards, and security considerations.

04

Approval & Merge

With passing CI and approvals secured, changes safely integrate into the codebase.

Learn more: Pull Request Workflow | Code Review Best Practices | GitHub PR Tutorial

Code Safety & Security Automation

Pre-Commit Hooks

Prevent passwords, API keys, PII, and destructive commands before they enter the repository. First line of defence.

Automated Linters

Enforce coding standards and catch common errors automatically. Consistency across the entire codebase.

Static Analysis

Detect security vulnerabilities, code smells, and potential bugs before deployment to production environments.

Secret Scanners

Continuously monitor for exposed credentials. CI fails builds immediately upon detecting unsafe code patterns.

Learn more: Pre-commit Hooks | Automated Linting | Secret Scanning

Real-World Team Implementation

Consider a team of five engineers building Apache Airflow DAGs for data orchestration:

- Each DAG developed in isolated feature branches
- Merged to dev for integration testing
- Automated CI/CD validates linting, unit tests, and integration tests
- Secret scanning runs in GitHub Actions
- Merge approvals required from peers before production deployment
- Main branch protected with enforcement rules

Learn more: Team Workflow | CI/CD Pipeline | Airflow Deployment

This workflow enables parallel development whilst maintaining production stability and code quality.

Automating Server Updates

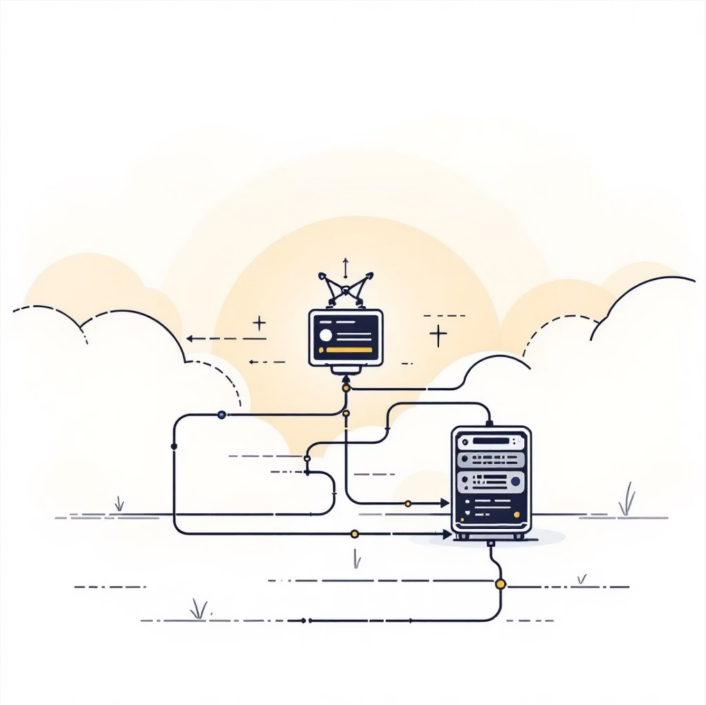
git-auto-deploy Solution

Manual server updates are error-prone. Automation ensures consistency and reliability:

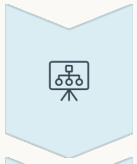
- Listens for GitHub/GitLab webhooks
- Automatically pulls repository updates
- Executes post-deployment scripts
- Supports branch-specific rules
- Logs all deployment events

Learn more: Git Auto Deploy GitHub | Automated Deployment | Webhook Setup

Install via pip, configure repositories in config.json, start the service, and set up webhooks. Updates flow automatically from commits to servers.



Deployment Automation Examples



Airflow DAG Updates

Automatically sync DAG repositories to production Airflow instances after dev branch merges complete.

Staging Environments

Update web applications on staging servers with every push for continuous testing and validation.

Container Rebuilds

Trigger Docker image rebuilds and service restarts automatically after main branch updates deploy.

Learn more: Deployment Automation | Auto Deploy Setup | Continuous Deployment

These automated workflows eliminate manual deployment steps, reduce human error, and accelerate delivery cycles.

Best Practices Summary

Structured Workflow

Feature → Dev → Main ensures safety whilst enabling parallel development across the entire team.

Protection & Automation

Branch protection plus CI/CD pipelines guarantee production code quality and stability.

Peer Reviews

Code reviews enforce quality standards, share knowledge, and reduce human errors significantly.

Security Tools

Automated scanning prevents secrets, PII, and destructive commands from entering repositories.

Server Automation

Automated Git pulls keep production and staging environments continuously synchronised and current.