Git

GitHub Documentation

Introduction

GitHub is a web-based platform for version control and collaboration using Git. It enables developers to manage repositories, track changes, and collaborate efficiently.

Key Features

- **Repositories**: Store and manage code projects.
- **Branches**: Create separate branches for different features or bug fixes.
- **Pull Requests (PRs)**: Propose and review code changes.
- **Issues**: Track bugs and feature requests.
- **Actions**: Automate workflows using CI/CD.
- **Wiki**: Maintain project documentation.
- **Security & Access Control**: Manage permissions and security vulnerabilities.

Getting Started

1. Creating a Repository

1. Sign in to GitHub.
2. Click the "+" icon and select "New repository."
3. Provide a name, description, and choose visibility (public/private).
4. Click "Create repository."

2. Cloning a Repository

```
git clone https://github.com/username/repository.git
```

3. Making Changes & Committing

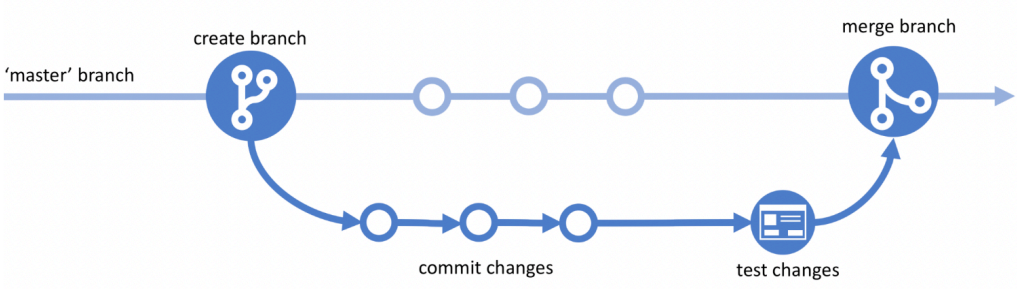
```
git add .
git commit -m "Commit message"
git push origin main
```

4. Creating a Pull Request

1. Push changes to a new branch.

2. Open the repository on GitHub.
3. Click "Compare & pull request."
4. Provide details and submit.

GitHub Flow



GitHub Flow is a simple and effective branching strategy designed for continuous delivery. It consists of the following steps:

1. **Create a branch**: Developers create a feature branch from `main`.
2. **Make changes and commit**: Work on the feature and commit changes frequently.
3. **Open a pull request (PR)**: Once the changes are ready, create a pull request to merge into `main`.
4. **Review and approve**: The team reviews the PR, provides feedback, and approves changes.
5. **Merge to main**: Once approved, the branch is merged into `main`.
6. **Deploy**: Since `main` is always deployable, changes can be automatically deployed.
7. **Delete the feature branch**: After merging, delete the branch to keep the repository clean.

GitHub Flow is particularly useful for projects that require frequent updates and deployments.

GitHub Actions (CI/CD)

GitHub Actions allows automation of tasks like testing and deployments. Example workflow:

```
name: CI
on: [push]
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Run tests
        run: npm test
```

Best Practices

1. Commit Best Practices

- Use clear and concise commit messages.
- Follow a consistent format (e.g., `feat: add new feature`, `fix: resolve issue`).
- Commit small, incremental changes frequently.

2. Branching Strategy

- Follow a standard branching strategy like **GitFlow** or **GitHub Flow**.
- Keep the `main` branch stable and protected.

- Use feature branches for new development and bug fixes.

3. Pull Request (PR) Best Practices

- Write clear PR descriptions explaining changes.
- Assign reviewers and request feedback early.
- Ensure PRs are small and focused to facilitate easier reviews.
- Use GitHub Actions to automate testing before merging.

4. Repository Security

- Enable **branch protection rules** to prevent accidental merges.
- Use **Code Scanning** to detect vulnerabilities.
- Require **signed commits** for verification.
- Use **two-factor authentication (2FA)** for better security.

5. Documentation & Collaboration

- Maintain a **README** file with project details and setup instructions.
- Use a **CONTRIBUTING.md** file to guide contributors.
- Keep **Issues** and **Discussions** well-organized.

Conclusion

GitHub enhances collaboration, version control, and CI/CD automation, making it an essential tool for modern development. By following best practices, teams can improve code quality, security, and workflow efficiency.