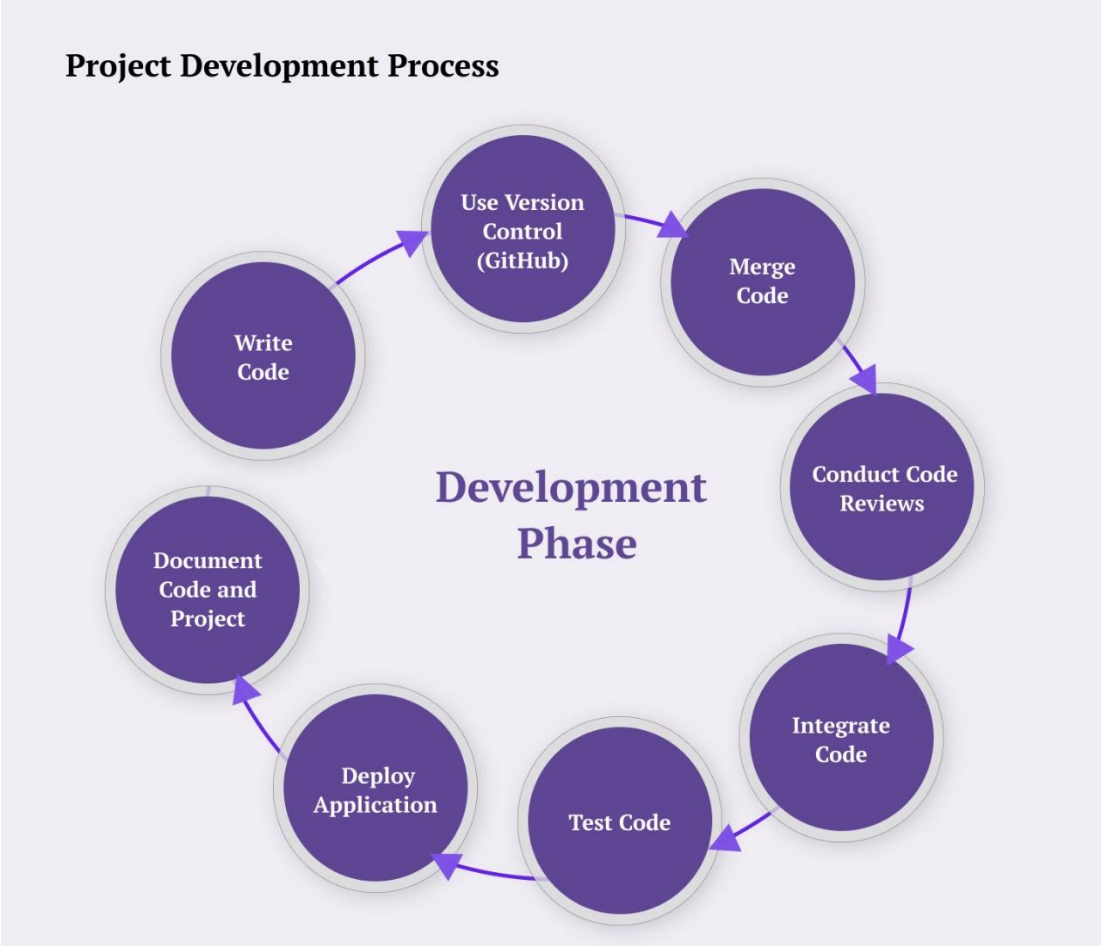
How Development Phase Works

Development Phase Detailed Documentation

The Development Phase is where the actual coding of the project occurs. It involves creating, testing, and integrating code to build the application according to the design specifications. This phase is critical for turning design plans into a functional product.



1. Code Implementation:

- **Writing Code**: Developers use Integrated Development Environments (IDEs) like Visual Studio Code or Sublime Text to write code based on the design documents.
- **Modular Development**: Code is often divided into modules or components, allowing multiple developers to work on different parts simultaneously.

2. Version Control:

- **Using GitHub**: GitHub manages code changes and versions. Developers commit code to branches and use pull requests for reviews and merging.
- **Branch Management**: Different branches are created for new features or bug fixes to prevent conflicts with the stable main branch.

3. Integration:

- **Merging Code**: Code from various branches is merged into the main branch. Conflicts may arise and need to be resolved during this process.
- **Continuous Integration**: Automated tools build and test code continuously to detect issues early in the development cycle.

4. Code Reviews:

- **Peer Review**: Team members review code changes to ensure quality and adherence to standards.
- **Feedback and Refinement**: Based on feedback, further refinements and bug fixes are made before the code is merged.

5. Documentation:

- **Code Comments**: Developers add comments to the code to explain its functionality and usage.
- **Project Documentation**: Comprehensive documentation helps in understanding the project structure and functionality.

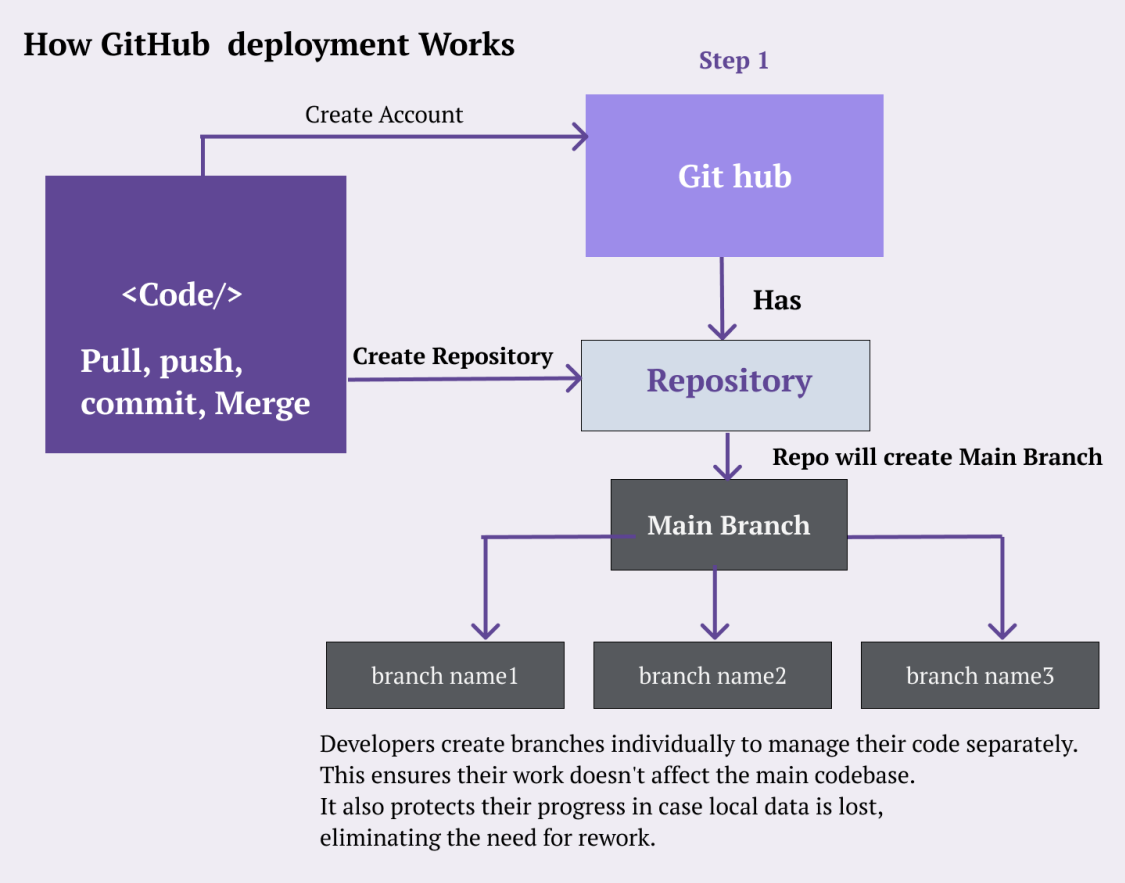
How GitHub process works

When developing an application, we use IDEs like VS Code or Sublime Text to work on individual modules. In a team, managing updates and preventing code conflicts can be challenging, especially if code is saved locally and accidentally deleted. GitHub, a DevOps version control platform, solves these issues by tracking changes, improving collaboration, and ensuring code is securely stored, helping teams work efficiently and avoid rework.

Using GitHub for Version Control Detailed Documentation

Introduction to GitHub

GitHub is a popular platform for managing and hosting code, built on Git for version control. It allows developers to track changes in their projects, collaborate with others, and keep the code organized. With GitHub, you can create branches for different features, review and merge code, and roll back to previous versions if needed. It also offers tools for issue tracking, continuous integration, and deployment, making it easier to manage and deliver your project efficiently.



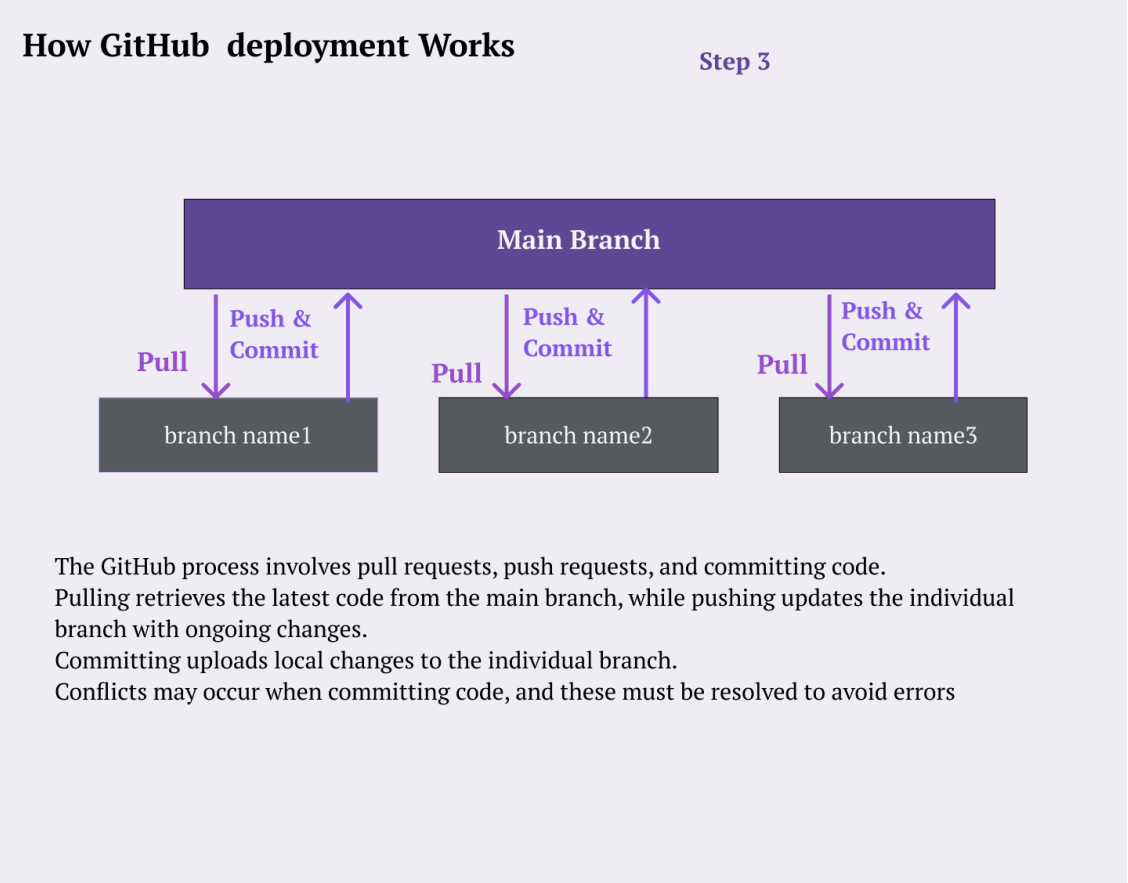
How to Create a Repository on GitHub

1. **Log In to GitHub:**
   Go to GitHub and log in to your account. If you don't have an account, you will need to sign up.
2. **Create a New Repository:**
   - Click the **"+"** icon in the upper right corner of the GitHub homepage.
   - Select **"New repository"** from the dropdown menu.
3. **Fill Out Repository Details:**
   - **Repository Name:** Enter a name for your repository.
   - **Description (Optional):** Provide a brief description of your project.
   - **Visibility:** Choose between **Public** (visible to everyone) or **Private** (visible only to you and collaborators).
   - **Initialize This Repository With:** You can optionally add a **README file**, a **.gitignore file** (to exclude certain files from being tracked), and choose a **license** for your repository.
4. **Create Repository:**
   - Click the **"Create repository"** button at the bottom of the page.

- By default, the Repository will create main branch (In older repositories, this was called the **master** branch.)

5. **Clone the Repository (Optional):**
   To work with the repository locally, copy the repository URL provided on the repository page and use the following command in your terminal:

How New Branch works After Cloning:



1. **Clone the Repository:**
   - In Visual Studio, go to the "Git" menu and select "Clone Repository."
   - Paste the repository URL from GitHub and choose a local path to store the repository.
   - Click "Clone." Visual Studio will download the project to your local machine.
2. **Open the Git Branches Window:**
   - Once the project is cloned, go to View > Git Repository Window (or click on the "Git" icon on the bottom-right toolbar).
   - You will see a list of branches in the "Git Repository" window, showing the current branch (usually the main branch).
3. **Create a New Branch:**
   - Right-click on the main branch (or any branch you want to base your new branch on).

- Select "New Local Branch from..." or click "Create New Branch" in the top options.
- Name your new branch (e.g., feature-xyz) and click Create.

4. **Switch to the New Branch:**
   - After creating the branch, you will automatically be switched to the new branch.
   - You can check the current branch in the bottom-right corner of Visual Studio or in the Git Repository window.
5. **Push the Branch to GitHub:**
   - After making changes, go to the Git menu and select "Push."
   - The new branch will be pushed to GitHub, and you can see it listed under your repository's branches.

By using Visual Studio's integrated Git tools, you can easily manage branches without needing to use the terminal.

Handling Conflicts and Merging Tasks in Vs Code

Visual Studio Code (VS Code) provides a user-friendly interface for handling Git operations, including resolving conflicts and merging branches. Here's a step-by-step guide on how to manage conflicts and merge tasks using VS Code:

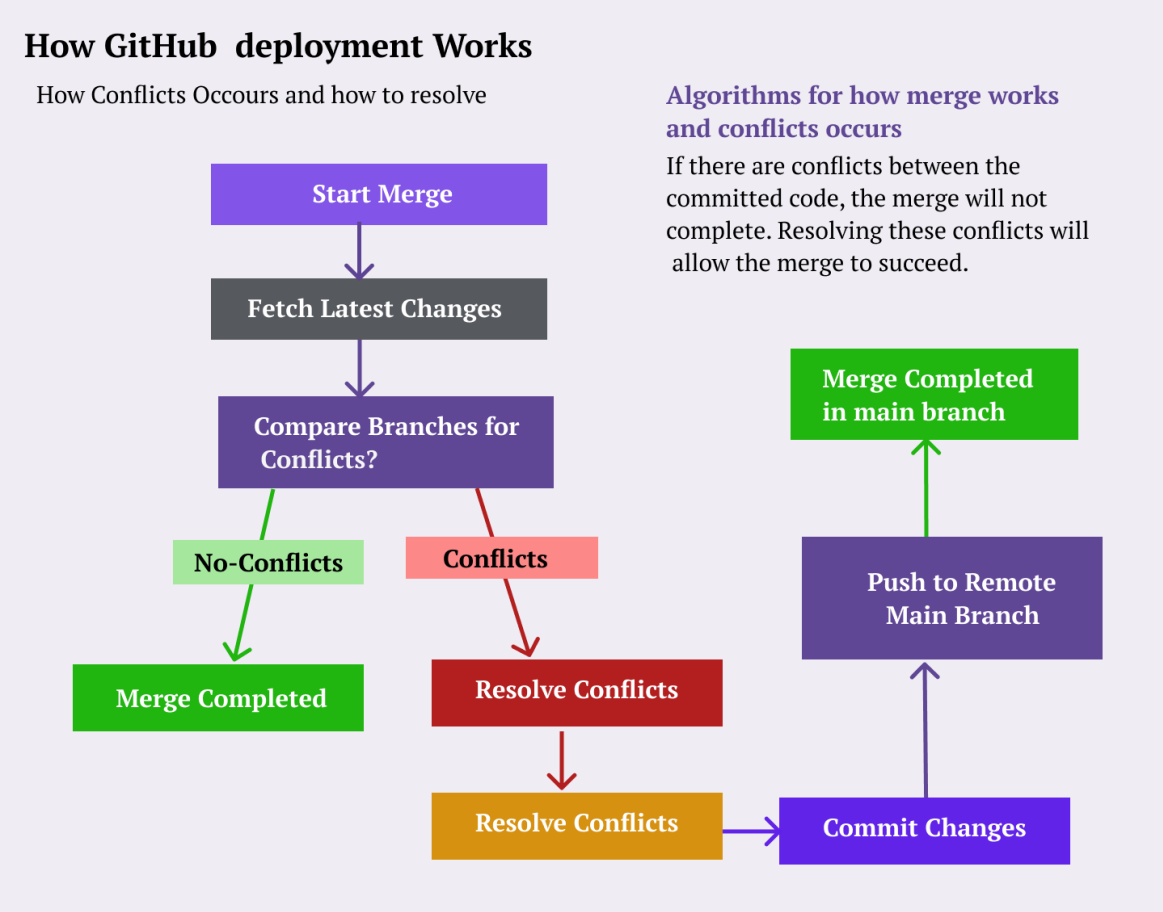
1. Identifying Conflicts

Conflicts arise during merging when:

- Two branches have changes to the same line of code.
- One branch deletes a file that another branch has modified.

In VS Code:

- Conflicts are indicated in the Source Control panel.
- Affected files will show a "U" status for unmerged files.

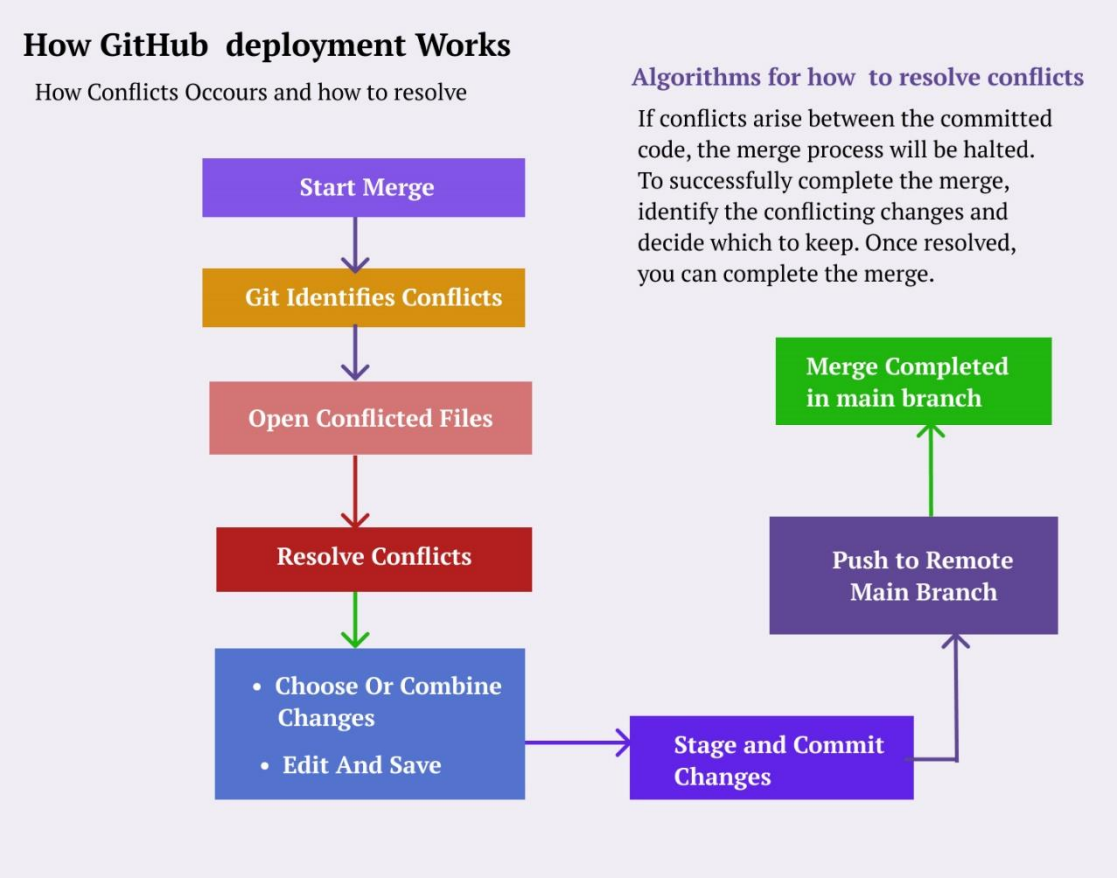


2. Resolving Conflicts

1. **Open the Conflicted File:**
   - Click on the file in the Source Control panel to view the conflict.
2. **View Conflict Markers:**
   - Conflict markers in the file show the differences:
     - <<<<<<< HEAD: Changes in your current branch.
     - =======: Separator between conflicting changes.
     - >>>>>>> branch-name: Changes from the branch being merged.
3. **Resolve Conflicts:**
   - Use VS Code's built-in options:
     - Accept Current Change: Keeps your changes.
     - Accept Incoming Change: Keeps the changes from the other branch.
     - Accept Both Changes: Combines both sets of changes.
     - Compare Changes: View differences side-by-side to make a decision.
4. **Finalize and Save:**

- Manually edit if needed to resolve conflicts.
- Remove conflict markers and save the file.

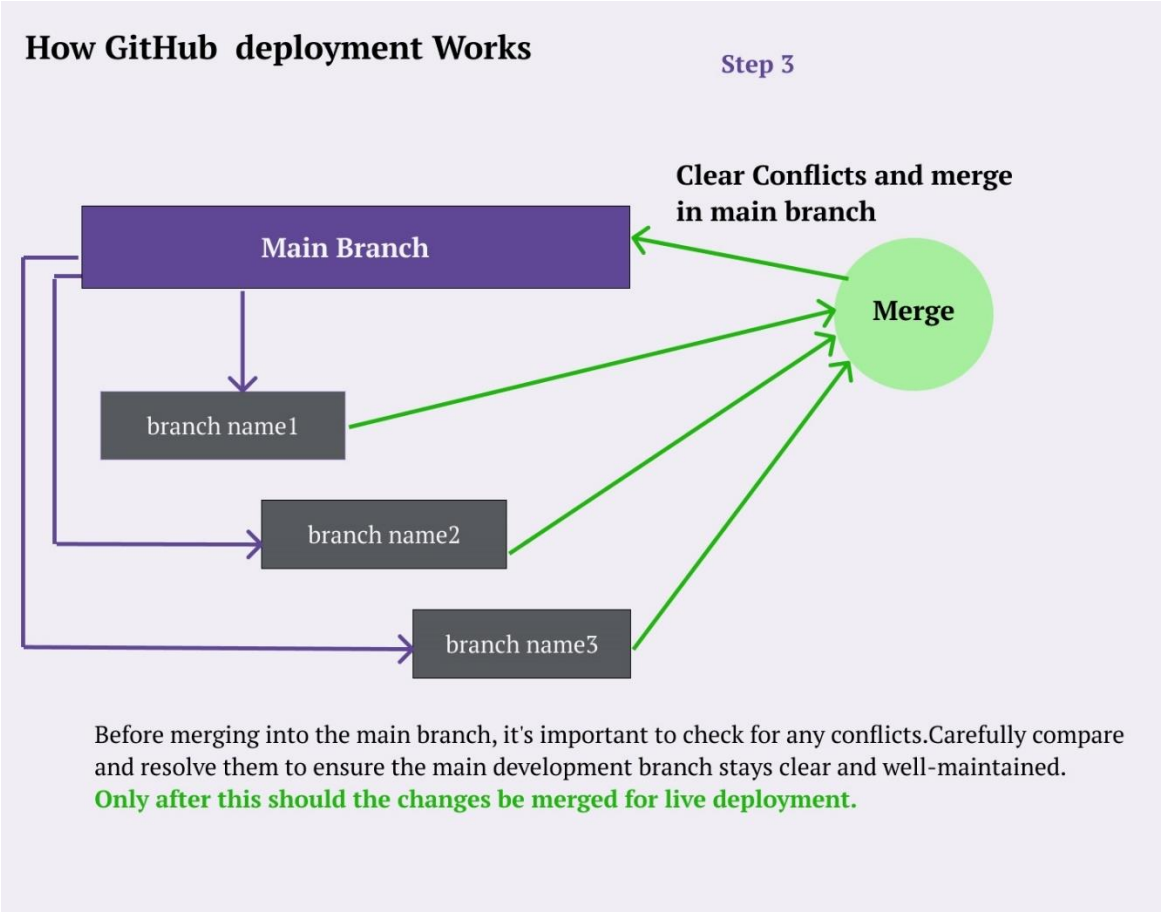
5. **Stage Resolved Files:**
   - In the Source Control panel, click the plus icon next to each resolved file or use Stage All Changes.
6. **Commit Changes:**
   - Write a commit message explaining the resolution.
   - Click the check mark to commit the changes.



3. Merging Tasks into the Main Branch

1. **Switch to Main Branch:**
   - Click the branch name in the bottom-left corner of VS Code.
   - Select main (or master if that is the default branch).
2. **Pull Latest Changes:**
   - Ensure main is up to date by pulling the latest changes.
   - Click the ... menu in the Source Control panel and select Pull.

3. **Merge Feature Branch:**
    - Click the ... menu again, select Branch, then Merge Branch.
    - Choose the branch to merge (e.g., feature-task).
4. **Resolve Additional Conflicts:**
    - If new conflicts arise, follow the same conflict resolution steps.
5. **Push Changes:**
    - Click the ... menu and select Push to upload the updated main branch to GitHub.



Basic GitHub Commands to Deploy Code

1. **git init**
   Initialize a new Git repository in your project folder.
2. **git add .**
   Add all your project files to the staging area, preparing them for commit.
3. **git commit -m "Your message"**
   Commit the staged changes with a descriptive message.
4. **git remote add origin <repository-URL>**
   Link your local repository to a remote GitHub repository.

5. **git push origin master**
   Push your committed changes to the master branch of the remote repository.
6. **git pull origin master**
   Fetch and merge changes from the remote repository to your local code.

These commands help you manage, track, and deploy your code to GitHub.

Conclusion

GitHub streamlines the development process by allowing teams to manage code efficiently, track changes, and collaborate seamlessly. With GitHub, you can create repositories, manage branches, and resolve conflicts, all while ensuring code security and organization. Visual Studio Code simplifies these tasks with its integrated Git tools. Using these resources effectively enhances collaboration, minimizes conflicts, and ensures smooth project deployment.