Introduction and Context

Modern web apps often offer light and dark themes to improve usability at different times of day. GitHub enables multiple developers to work on the same codebase without stepping on each other's toes by using branches, pull requests, and merge workflows. By the end of this tutorial, you'll understand how to create feature/fix branches, merge with fast-forward and squash strategies, manage draft versus standard pull requests, resolve conflicts manually, and synchronize local work with the remote repository.

Case Study Overview

Problem Statement
BrightUI maintains a public website. Designers have requested a Dark Mode toggle. Two front-end engineers, Jamie and Priya, will each implement parts of the Dark Mode CSS on separate branches. They must collaborate without overwriting one another, review each other's work, handle any merge conflicts when both edit the same CSS file, and merge into `main` using both fast-forward and squash strategies. Success means shipping Dark Mode with a clean commit history and no lost changes.

Learning Objectives

- Create and use feature branches for isolated development
- Merge branches with fast-forward and squash techniques
- Open draft and standard pull requests and conduct reviews
- Identify and resolve merge conflicts manually
- Keep local branches synchronized via pull, push, and fetch workflows

Concepts Explained with Analogies

Branching
Analogy: Working on a photocopy of a blueprint so you can sketch changes without altering the original drawing.

Technical: A branch is a pointer to a series of commits. Creating a feature branch isolates work until it's ready to merge back into `main`

Merging

Analogy: Gluing your sketch back onto the master blueprint, either by appending your changes (fast-forward) or by consolidating your entire sketch into one neat overview (squash).

Technical: A fast-forward merge moves the base branch pointer forward when no divergent commits exist, while a squash merge condenses all feature-branch commits into a single commit on the target branch

Pull Requests

Analogy: Passing your sketch to a peer for review, initially as a draft for early feedback, then as a final version for formal approval.

Technical: Draft pull requests signal "work in progress," preventing accidental merges, whereas standard pull requests are ready for review and merging

Merge Conflicts

Analogy: Two artists coloring the same area of a painting in different hues-you must decide which stroke to keep.

Technical: When Git cannot reconcile competing edits on the same lines, a conflict arises; resolving it requires manual editing and committing the resolution

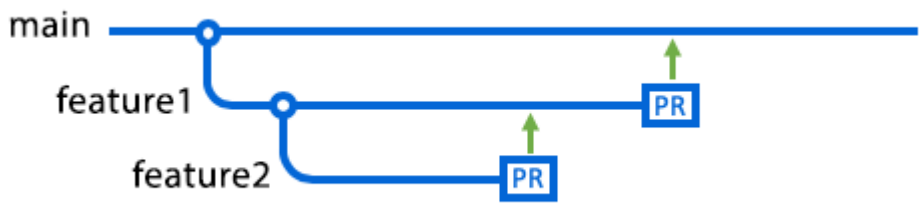
Syncing

Analogy: Checking if everyone's blueprints are up to date before you start drawing.

Technical: Pulling fetches and merges remote changes, pushing shares your commits, and fetch-only lets you inspect incoming changes before integrating them

Visualization of branching and merging

Here someone has created a branch called `feature1` from the `main` branch, and you've then created a branch called `feature2` from `feature1`. There are open pull requests for both branches. The arrows indicate the current base branch for each pull request. At this point, `feature1` is the base branch for `feature2`. If the pull request for `feature2` is merged now, the `feature2` branch will be merged into `feature1`.



In the next diagram, someone has merged the pull request for `feature1` into the `main` branch, and they have deleted the `feature1` branch. As a result, GitHub has automatically retargeted the pull request for `feature2` so that its base branch is now `main`.

Now when you merge the `feature2` pull request, it'll be merged into the `main` branch.

Step-by-Step Guided Walkthrough

Step 1: Clone and Create Feature Branches

```
git clone https://github.com/BrightUI/website.git     # Clone the main repo
cd website
git checkout -b feature/dark-mode-toggle              # Jamie's branch
```

Checkpoint: Why avoid committing directly to `main`?

Meanwhile, Priya runs:

```
git fetch origin
git checkout -b feature/dark-mode-colors origin/main  # Priya's branch
```

Step 2: Develop Dark Mode CSS and Commit

Jamie edits `styles/theme.css`:

```
/* Add Dark Mode toggle */
body.dark-mode { background: #121212; color: #e0e0e0; }
```

```
git add styles/theme.css
git commit -m "feat: add dark-mode toggle class"      # Descriptive commit
git push -u origin feature/dark-mode-toggle
```

Priya edits the same file to add color variables:

```
:root  {  --bg-dark: #121212;  --text-light: #e0e0e0;  }
```

```
git add styles/theme.css
git commit -m "feat: define dark mode CSS variables"
git push -u origin feature/dark-mode-colors
```

Step 3: Open Draft Pull Requests

Jamie (Draft PR):

```
gh pr create --draft --base main --head feature/dark-mode-toggle \
  --title "WIP: Dark Mode Toggle" \
  --body "Initial toggle implementation; needs CSS refinements"[2]
```

Priya (Standard PR via UI):

- Navigate to **Pull requests** → **New pull request**
- Set **base**: main, **compare**: feature/dark-mode-colors
- Enter title “Add dark mode color variables” and description
- Click **Create pull request**

Checkpoint: When should you mark a PR as draft versus ready?

Step 4: Review and Resolve Merge Conflicts

- Both PRs touch `styles/theme.css` -a conflict arises when merging Jamie’s completed PR first.
- On GitHub, click **Merge pull request** → conflict alert appears4.

- Click **Resolve conflicts**. The editor shows:

```
<<<<<<< HEAD
body.dark-mode { background: #121212; color: #e0e0e0; }
=======
:root { --bg-dark: #121212; --text-light: #e0e0e0; }
>>>>>>> feature/dark-mode-colors
```

- Manually combine:

```
:root { --bg-dark: #121212; --text-light: #e0e0e0; }
body.dark-mode { background: var(--bg-dark); color: var(--text-light);
```

- Click **Mark as resolved** and **Commit merge**.

Checkpoint: Why is manual conflict resolution necessary rather than automatic?

Step 5: Merge Strategies

Fast-Forward Merge (for Jamie's toggle branch if no conflicts):

```
git checkout main
git pull origin main
git merge --ff-only feature/dark-mode-toggle      # Moves main forward[8]
git push
```

Squash Merge (for Priya's combined CSS changes):

```
gh pr merge feature/dark-mode-colors --squash \
  --title "feat: implement dark mode styling" \
  --delete-branch                             # Single commit on main[8]
```

Step 6: Sync Local Branches

```
git checkout main
git pull --rebase origin main                  # Rebase for cleaner history
git checkout feature/dark-mode-toggle
git pull --ff-only                              # Fast-forward to updated m
```

Checkpoint: What's the difference between `git pull` and `git fetch`?

Best Practices and Tips

- Use **descriptive branch names** (`feature/`, `fix/`) to clarify purpose.
- Open **draft PRs** early to share work-in-progress and gather initial feedback.
- Prefer **fast-forward** mergers for short-lived branches to avoid unnecessary merge commits.
- Use **squash merges** for feature branches with many small commits to keep `main` history tidy.
- Frequently **sync** branches with `main` to reduce conflict scope.
- Review diffs carefully and **resolve conflicts manually** to ensure intended code remains.