Expert Cloud Consulting
Enhance Optimise & Scale
ASCP GPUonCLOUD Pvt Ltd

"Expert Cloud Consulting" -

Documentation for GiHub Branching Strategy[Title,18, Arial]

14.April.2025 [Subtitle,14, Arial]

version 1.0

—

Contributed by Yogiraj Deshpande [Normal text,14, Arial]
Approved by Akshay Shinde(In Review)
Expert Cloud Consulting
Office #811, Gera Imperium Rise,
Hinjewadi Phase-II Rd, Pune, India – 411057

“Expert Cloud Consulting”
Branching Strategy & Development Workflow Documentation [Title,18, Arial]

1.0: Document Overview:

This document outlines the branching strategy and development workflow followed during the local repository setup and deployment to an AWS instance. The aim is to maintain a clean, organized codebase and follow basic collaboration practices using Git and GitHub, even without advanced automation like CI/CD

2.0: Objective:

- Use Git branches effectively to manage features, fixes, and production-ready code.
- Simulate a professional development workflow using GitHub, even for individual developers or small teams.
- Deploy code manually from local branches to a single AWS instance.

3.0: Prerequisites:

The following practices/tools are required for this workflow:

- Git installed on local machine
- GitHub account with a repository
- Access to a single AWS EC2 instance (for manual deployment)
- Basic knowledge of Git commands and GitHub pull request

4.0: Branch Types & Naming Conventions:

In our Git workflow, we use several distinct branch types to organize and structure development. The naming conventions for each type ensure clarity and consistency.

Branch Types:

Branch Name	Description	Examples
main	Stable production-ready code	main
develop	Integration branch for ongoing development	develop
feature/*	Individual feature development branch	feature/user-login
bugfix/*	Fixes to non-critical issues	bugfix/button-align
hotfix/*	Critical fixes for production	hotfix/api-crash

5.0: Development Workflow:

5.1: Local Development Process:

1. Clone the repository:

```
git clone <repo-url>
cd <repo-name>
```

- **Why?** Cloning creates a local copy of the GitHub repository, allowing development to begin.

2. Create `main` and `develop` Branches:

```
# Start from main
git checkout -b main
git commit --allow-empty -m "Initial commit on main"
git push -u origin main

# Create develop branch from main
git checkout -b develop
git push -u origin develop
```

- **Why?** `main` holds production code. `develop` is where new work is integrated. `--allow-empty` creates a branch with an initial empty commit.

3. Create a Feature Branch:

```
# From develop
git checkout develop


# Create and switch to new feature branch
git checkout -b feature/user-login

# Make changes, then commit
git add .
git commit -m "Add login feature"

# Push to GitHub
git push origin feature/user-login
```

- **Why?** Feature branches isolate development of new features and protect the stability of `develop`.

4. Create a Bugfix Branch:

```
# From develop
git checkout develop

# Create and switch to bugfix branch
git checkout -b bugfix/navbar-alignment

# Apply fix, commit, and push
git add .
git commit -m "Fix navbar alignment"
git push origin bugfix/navbar-alignment
```

- **Why?** Bugfix branches are used to resolve minor issues during development without affecting ongoing feature work.

5. Create a Hotfix Branch:

```
# From main
git checkout main

# Create hotfix branch
git checkout -b hotfix/payment-crash

# Apply fix, commit, and push
git add .
git commit -m "Fix payment crash issue"
git push origin hotfix/payment-crash
```

- **Why?** Hotfixes are for critical issues found in production and must be resolved directly from `main`

6. Open a Pull Request (PR): **What is a PR?** A Pull Request is a GitHub feature that allows developers to propose changes from one branch to another, usually from a feature branch to `develop` or `main`.

- **Why?**
- To enable code reviews
- To track changes and provide visibility
- To simulate team collaboration
- **How to Use a PR:**

1. Push your feature or fix branch to GitHub.
2. Navigate to the repository on GitHub.
3. GitHub will prompt: “Compare & pull request” for the newly pushed branch.
4. Click on it and select the base branch (`develop` or `main`) and compare branch (`feature/*`, `bugfix/*`, or `hotfix/*`).
5. Add a meaningful title and description of the changes.
6. Click “Create pull request.”
7. Team members (or you, if solo) can review the changes, leave comments, and approve.
8. Once approved or verified, click “Merge pull request.”

6.0: Pull Requests & Approvals:

- PRs are created to simulate collaboration and team review.
- While formal approvals may not be required, comments are added to simulate feedback.
- After validation, changes are merged into the `develop` branch.

7.0: Conflict Resolution:

When a conflict occurs while merging, the following steps are taken:

```
git checkout feature/some-feature
git pull origin develop
# Resolve the conflicts in indicated files

git add .
git commit -m "Resolved merge conflicts with develop"
git push
```

After resolving conflicts and updating the feature branch, the branch is merged into `develop`.

Example Scenario:

9. Two developers (or the same developer in different branches) modify the same line in `header.html`:
   - `feature/add-logo` adds a new logo tag
   - `feature/change-title` modifies the page title

1. When both branches are merged into `develop`, Git detects a conflict in `header.html`
2. Developer working on `feature/change-title` runs:

```
git checkout feature/change-title
git pull origin develop
```

3. Git shows a conflict in `header.html`. The file looks like:

```
<<<<<<< HEAD
<title>New Title</title>
=======
<img src="logo.png" alt="Logo">
>>>>>>> develop
```

4. Developer manually edits the file to include both changes:

```
<img src="logo.png" alt="Logo">
<title>New Title</title>
```

5. Then runs:

```
git add header.html
git commit -m "Resolve conflict in header.html by combining logo and
new title"
git push
```

6. The PR is then updated and merged to `develop` successfully.

8.0: Release to Production:

Once all changes are reviewed and merged into `develop`, the release process is initiated:

```
git checkout main
git merge develop
git push origin main
```

Why? This brings all new changes from `develop` into `main` for deployment.

9.0: Manual Deployment to AWS:

Once code was merged to `main`, deployment was done manually:

```
# Connect to EC2
ssh ubuntu@<public-ip>

# Navigate to project directory and pull latest code
cd /var/www/project
git pull origin main

# Restart web server if needed
sudo systemctl restart apache2
```

Why? This method allows for simple, secure deployment to a single server without using automation tools.

10.0: Summary:

- This branching strategy ensures clean separation between development and production.
- Pull Requests enable code review and simulate team collaboration.
- Conflict resolution practices maintain code integrity during merges.
- Manual deployment to a single AWS instance is achieved directly from the `main` branch using SSH and Git.