Branch Management

Listing Branches

- Command: git branch
- Output example

$ git branch

* iss53*

** master*

* testing*

- The * symbol shows the current branch you're working on (HEAD).

Viewing Last Commit on Each Branch

- Command: git branch –v
- Output Example:

$ git branch -v

iss53 93b412c Fix javascript issue

** master 7a98805 Merge branch 'iss53'*

testing 782fd34 Add scott to the author list in the readme

- Shows the last commit on each branch for easy tracking.

Branch Merging Status

- Command: git branch –merged
- Output Example:

$ git branch --merged

iss53

**Master*

- Lists branches merged into the current branch.

Checking Unmerged Branches

- Command: git branch --no-merged
- Output Example:

$ git branch --no-merged

testing

- Shows branches not yet merged into the current branch.

Deleting Branches

- Command for safe deletion: git branch -d <branch-name>
- Output example:

$ git branch -d testing

error: The branch 'testing' is not fully merged.

- Command for force deletion: git branch -D <branch-name>

Renaming Branches Locally

- Command: git branch --move old-name new-name
- Output Example:

$ git branch --move bad-branch-name corrected-branch-name

- Renames the branch locally.

Renaming Branches on Remote

- Command to push renamed branch: *$ git push --set-upstream origin corrected-branch-name*

- Command to delete old branch: *$ git push origin --delete bad-branch-name*

Changing the Master Branch Name

- Local rename*: git branch --move master main*
- Remote rename: *git push --set-upstream origin main*

Deleting the Old Master Branch

- Command: *git push origin --delete master*
- Delete after ensuring all configurations and scripts are updated.

Branching Workflows

- It is a robust branching model for Git.
- It is designed to manage development and release cycles in a more structured way.
- Git Flow introduces the concept of multiple branches for managing different aspects of the development process: feature development, releases, and hotfixes.

Branches

- There are two long-lived branches that form the backbone of the Git Flow model:
  1. master(main)
  2. develop

Main/master Branch

- This branch holds the production-ready code at all times.
- Each commit in this branch represents a stable, production release. In other words, whenever code is merged into master, it’s ready to be deployed to production.
- Only release commits and hotfixes are merged here.

Develop

- The develop branch is where all ongoing development happens. It represents the latest state of development.
- Feature branches are merged into develop, and the code is tested here before being prepared for a production release.
- When a release is ready, develop is merged into master.

Other supporting branches

- Feature Branches
- Release Branches
- Hotfix Branches

Feature Branches (feature/*)

- Feature branches are used to develop new features or functionality.
- Developers create a new feature branch when starting work on a specific feature, allowing them to work in isolation without affecting the main codebase.
- Created from: develop
- Merged back into: develop
- Naming convention: feature/feature-name

- Create a new feature branch

git checkout develop git checkout -b feature/new-feature

- Work on the feature, then merge back

git checkout develop git merge feature/new-feature

- Optionally delete the feature branch

git branch -d feature/new-feature

Release branches(release/*)

- Release branches help prepare for an upcoming release.
- When develop has reached a stable point and is ready for release, a release branch is created.
- In this branch, final adjustments like minor bug fixes, documentation updates, or version number increments are made.
- Once the release is finalized, the release branch is merged into both master and develop.
- Merging into master means the code is now production-ready.
- Merging into develop ensures that any changes made during the release process (like bug fixes) are also reflected in ongoing development.

- Created from: develop
- Merged into: master and develop
- Naming convention: release/version-number

- Create a release branch from develop *:* *git checkout develop*
  git checkout -b release/1.0.0

- Make final adjustments, then merge into master and develop: *git checkout master*
  git merge release/1.0.0
  git checkout develop
  git merge release/1.0.0
- Tag the release for reference: *git tag -a 1.0.0 -m "Release version 1.0.0"*
- Delete the release branch : *git branch -d release/1.0.0*

Hotfix(hotfix/*)

- Hotfix branches are used to address urgent issues or bugs found in production.
- They are created directly from master, allowing you to work on fixing the bug without affecting other ongoing development.
- Once the issue is fixed, the hotfix is merged into both master (to deploy the fix to production) and develop (so that ongoing development also contains the fix).
- Created from: master
- Merged into: master and develop
- Naming convention: hotfix/issue-description

- Create a hotfix branch from master

  git checkout master

  git checkout -b hotfix/critical-bug

- Fix the bug, then merge into master and develop

  git checkout master

  git merge hotfix/critical-bug

  git checkout develop

  git merge hotfix/critical-bug

- Tag the hotfix release

  git tag -a 1.0.1 -m "Hotfix for critical bug"

- Delete the hotfix branch

  git branch -d hotfix/critical-bug

Advantages of Git Flow

- **Clear Structure:** Git Flow provides a well-defined branching model, which ensures that the development, release, and bug-fixing processes are organized and traceable.
- **Parallel Development:** By isolating features and bug fixes in branches, multiple team members can work simultaneously without stepping on each other’s toes.
- **Stability in Production:** The separation of master (production) and develop (ongoing development) ensures that only fully tested and stable code makes it into production.
- **Flexibility:** The model accommodates multiple releases, features, and emergency bug fixes without disruption.

Challenges of Git Flow

- **Complexity for Small Teams:**
- **Continuous Deployment:** Git flow doesn’t align with the concept of continuous development. It encourages long-running feature branches and formal release stages.