Git & GitHub

Phoenix Programming

What is GitHub?

GitHub is a cloud-based platform built on Git that allows developers to store, manage, and share code, enabling real-time, collaborative software development and open-source projects. It provides version control to track changes, allows teams to work on different parts of a project simultaneously using branches, and offers tools for bug tracking, project management, and code review.

Git Terminology (Part 1)

- **Repository (Repo):** The centralized location where all project files and their revision history are stored. This can be a local repository on a personal machine or a remote repository hosted on a server (e.g., GitHub).
- **Commit:** A snapshot of changes made to the project files at a specific point in time. Each commit has a unique identifier (hash) and includes a message describing the changes.
- **Branch:** A separate line of development within a repository. Branches allow developers to work on new features or bug fixes without affecting the main codebase.
- **Main/Master:** The default or primary branch in a Git repository, conventionally where the stable version of the project resides.
- **Head:** A pointer to the current commit in the currently checked-out branch.
- **Origin:** The conventional name for the primary remote repository from which a local repository was cloned.
- **Staging Area (Index):** An intermediate area where changes are prepared before being committed. Files are added to the staging area using git add.

Git Terminology (Part 2)

- **Clone:** To create a local copy of a remote repository.
- **Fetch:** To download changes from a remote repository to your local repository without merging them into your current branch.
- **Pull:** To download changes from a remote repository and automatically merge them into your current local branch (git fetch followed by git merge).
- **Push:** To upload local commits to a remote repository.
- **Merge:** To combine the changes from one branch into another.
- **Squash:** To combine the changes from one branch into a single commit on another branch.
- **Rebase:** To move or combine a sequence of commits to a new base commit, effectively rewriting commit history.
- **Pull Request (PR):** A proposal to merge changes from one branch into another, typically used in remote repositories for code review.
- **Fork:** To create a personal copy of a repository, often used when contributing to open-source projects.

Basic Git CLI Commands (Part 1)

Repository Management:

- **git init:** Initializes a new Git repository in the current directory.
- **git clone <*repository-url*>:** Creates a local copy of an existing remote repository.

Making and Saving Changes:

- **git status:** Displays the state of the working directory and the staging area, showing modified, staged, and untracked files.
- **git add <*file*>:** Adds changes from the working directory of a specific file to the staging area. Use git add . to stage all changes.
- **git commit -m *"Commit message"*:** Records the staged changes in the repository's history with a descriptive message.

Basic Git CLI Commands (Part 2)

Branching and Merging:

- **git branch:** Lists all local branches in the repository.
- **git branch <*branch-name*>:** Creates a new branch.
- **git checkout <*branch-name*>:** Switches to a different branch.
- **git merge <*branch-name*>:** Merges changes from the specified branch into the current branch.

Remote Repository Interaction:

- **git pull:** Fetches changes from a remote repository and merges them into the current local branch.
- **git push origin <*branch-name*>:** Uploads local commits to the specified remote branch.

Viewing History:

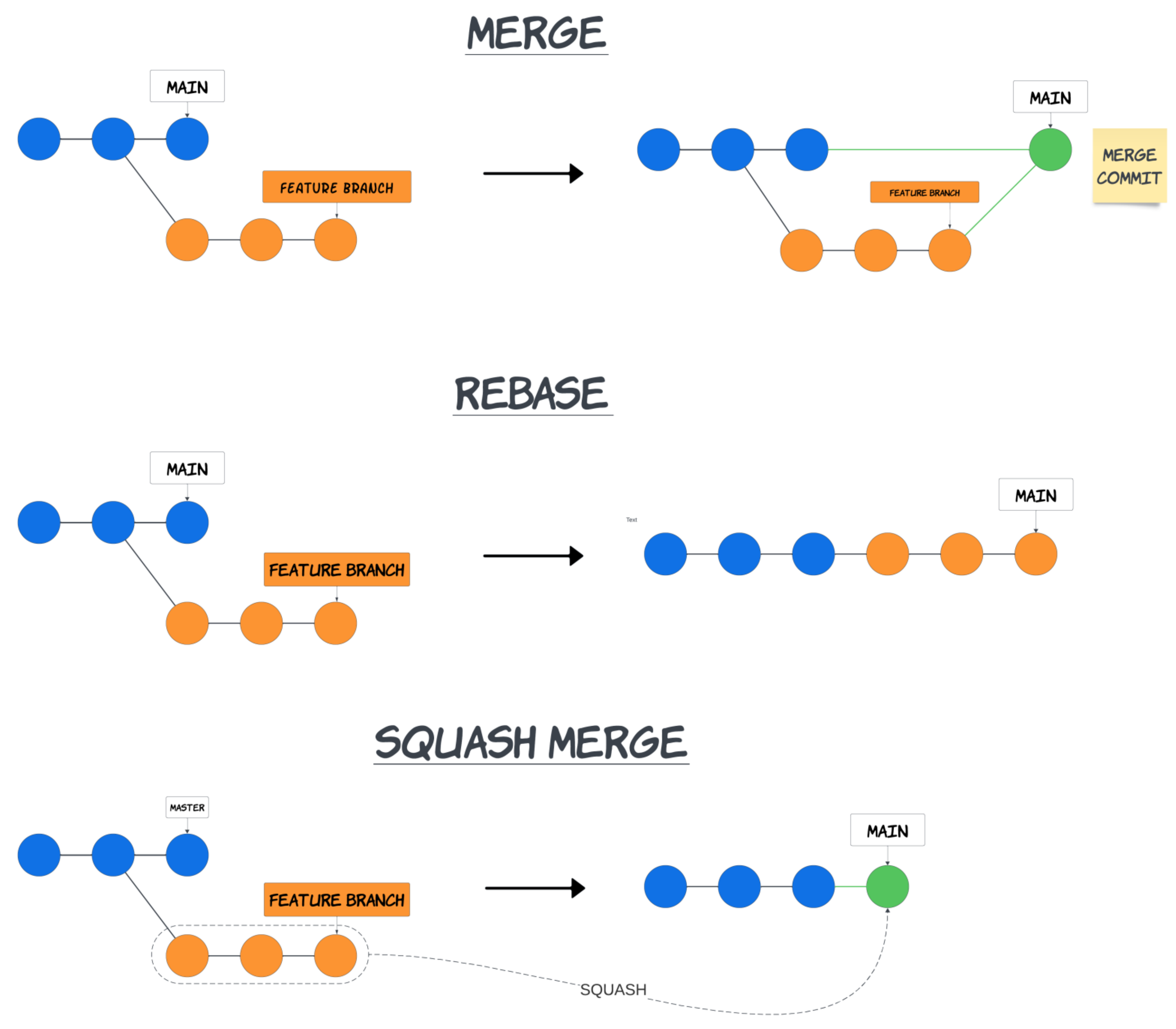
- **git log:** Shows the commit history for the current branch.

What is a Pull Request (PR)?

A pull request is a formal proposal in a software development workflow to merge changes made in one branch into another. It acts as a request for the project's maintainers to review and potentially incorporate new code or modifications into the main codebase after a developer has made changes in a separate branch. This process allows for code review, collaboration, and feedback before the changes are accepted, helping to maintain code quality and prevent unintended issues.

Merge vs. Rebase vs. Squash Commits

- **Merge:** Creates a merge commit, preserving the full history, creating a non-linear graph.
- **Rebase:** Moves commits to a new base, reapplying them to create a linear, cleaner history without extra merge commits.
- **Squash:** Condenses all feature branch commits into a single commit, losing individual history for a cleaner main branch.



PR Conversations and Resolving Them

- Pull Requests (PRs) request to merge implemented code into the main branch
- PRs contain a title field and a description field
- PRs must be approved by other code contributors
- PRs should be created when a feature branch has been implemented and thoroughly tested

How to Review a PR

- Ensure that the committed code has been tested and implemented correctly
- Provide feedback related to the changes (why you are approving it, or what must be done to make this PR approvable)
- PRs are a dynamic discussion, not just accept/reject

Our Standards

1. Never commit directly to main
2. Each task, feature, or bug needs to be on its own branch
3. Branch Naming Format: <First Name>_<Last Initial>_<Issue #>_<Task, Feature, or Bug Name> (for easy identification and consistency)
4. Don’t perform any merge commits, only squash or rebase commits (to maintain linear commit history)
5. Always merge the main branch into your branch and perform extensive testing before creating a pull request
6. 2 reviewers from the team are needed to approve each pull request merging into main
7. Resolve all conversations before completing a pull request merge
8. Always do a squash commit when merging your branch into main through a pull request
9. After your PR has successfully merged into main, delete the remote and local branches you created for that task, feature, or bug