Understanding Branches and Pipelines

Three core concepts

- **Branches:** Separate workspaces where developers can safely build and test new ideas.
- **Pipelines:** Automated quality inspectors that check the code for errors at every stage.
- **GitHub:** The central hub or repository where the official code is stored and managed.

1. The Key Landmarks: Defining Our Core Concepts

The Master Branch

At the heart of the GitHub repository is the Master branch. This is the official, stable, and working version of the software project. It represents the "source of truth"—the definitive version of the code that has been tested and approved.

Think of the Master branch as the master copy of a book in a library. It's the one that everyone reads and trusts to be the correct version. You don't write your rough drafts directly in it.

The Feature Branch

When a developer needs to work on a new feature or a bug fix, they don't work directly on Master. Instead, they Create a feature branch (like the one named “feature/**”). A feature branch is a temporary, independent copy of the Master branch.

The primary benefit is **isolation**. It allows a developer to make Changes and experiment freely without any risk of destabilizing the official Master branch.

The Pipeline

A Pipeline is an automated series of steps that build, test, and prepare code. The two distinct pipelines feature and master/main pipeline, each with a specific job that acts as an automated quality inspector.

Pipeline Type	Primary Job (Based on the Diagram)
Feature Pipeline	To check the new code on a feature branch by running `mvn clean verify` and ensuring it passes the team's definition of a "good test".
Master/Main Pipeline	To deploy the official, merged code by running `mvn clean deploy`, which results in a `Created artifact` for the `Remote Repository`.

2. The Development Journey: From Idea to Approved Feature

This part of the process focuses on developing a new feature in a controlled and safe environment.

Step 1: Starting the Work (`Create a feature branch`)

The journey begins when a developer creates a new `feature branch` (e.g., feature/test) from the `Master branch`. This action provides them with a clean, identical, and independent copy of the project to work in.

Step 2: The First Quality Check (`Feature Pipeline`)

As the developer makes `Changes` to the code on their feature branch, the `Feature Pipeline` is automatically triggered. This pipeline runs the "**`mvn clean verify`**" command, which is a powerful command that compiles the code and runs a suite of automated checks. Its goal is to confirm the new code passes what the team considers a "good test" before it can proceed, catching bugs or errors early.

Step 3: Getting the Green Light (`Success` and `Promote`)

The pipeline only proceeds to the `Promote` step **if it achieves `Success`**. This status is the automated green light indicating the code is stable and works as intended on the feature branch. The developer can now `Promote` their changes. This action represents proposing to merge the new feature into the `Master` branch, typically through a process called a "Pull Request," where other team members can review the code before it's approved.

With the feature built and successfully tested in isolation, it's now ready to be integrated into the main project.

3. The Final Step: Releasing the Official Version

This is the final stage where the approved feature becomes part of the official software.

Step 1: Merging into `Master`

Once the `Promote` action (or Pull Request) is reviewed and approved by the team, the code from the `feature branch` is officially merged into the `Master` branch. This updates the "source of truth" with the new, tested feature.

Step 2: The Final Production Line (`Master/Main Pipeline`)

This merges into `Master` immediately and automatically triggers the `Master/Main Pipeline`. This pipeline's job is to prepare the final, official version of the software for release. It runs the **"`mvn clean deploy`"** command, which builds the complete application and prepares it for users.

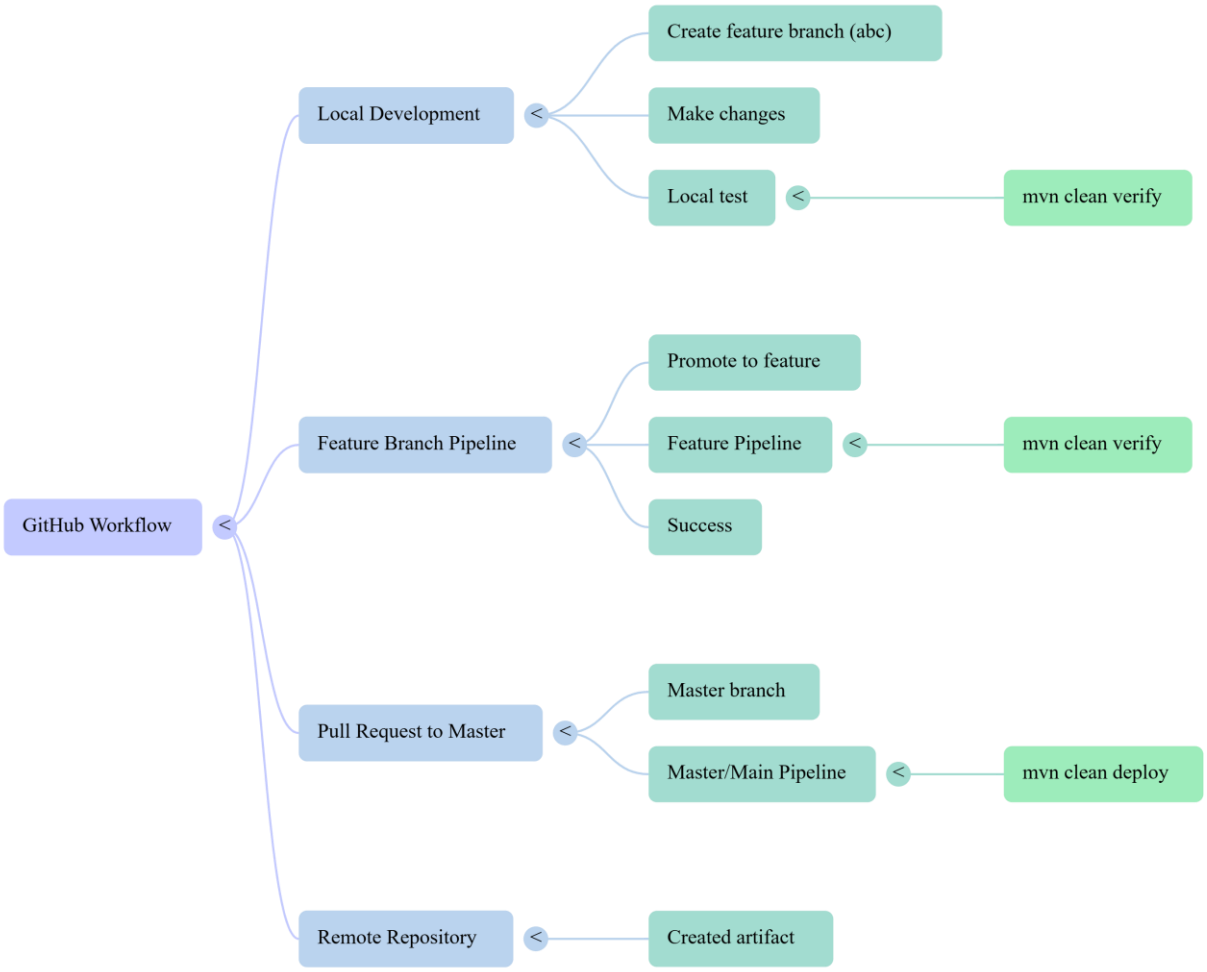
Step 3: The Finished Product (`Created artifact`)

The successful output of the `Master/Main Pipeline` is a `Created artifact`. An artifact is the compiled, packaged, and ready-to-use version of the application (e.g., a `.jar` or `.war` file). This finished product is then stored in a `Remote Repository`, where it can be accessed for deployment to live servers.

4. Conclusion: Why Work This Way?

Following this structured workflow of branches and pipelines provides several critical benefits for any software development team.

1. **Safety** By using feature branches, developers create an **isolated workspace** to make `Changes` without fear of destabilizing the `Master` branch. This prevents unstable or unfinished work from ever breaking the official version of the project.
2. **Quality** The automated pipelines act as tireless **gatekeepers**. The `Feature Pipeline` ensures every change passes a "good test" before it can even be proposed for merging, while the `Master/Main Pipeline` verifies the final product, catching bugs early and often.
3. **Collaboration** This branching model allows multiple developers to work in **parallel**, each on their own `feature branch`, without creating conflicts or overwriting each other's work. This dramatically improves a team's efficiency and speed.


GitHub Workflow
Local Development
Create feature branch (abc)
Make changes
Local test
mvn clean verify
Feature Branch Pipeline
Promote to feature
Feature Pipeline
mvn clean verify
Success
Pull Request to Master
Master branch
Master/Main Pipeline
mvn clean deploy
Remote Repository
Created artifact