Branching & Merging

AiCE Software Engineering Foundations — The Developer Environment

So far, every change you've made has gone straight onto a single timeline. That works for small, solo projects, but it breaks down the moment you want to try something risky without threatening a live, working system. This lesson introduces branches — parallel versions of your repository — and how to merge finished work back into the main line.

1. Understanding Branches

A branch is a parallel version of your repository. It lets you work on new features, fix bugs, or experiment — all without touching the main, stable version of your project.

Imagine you're building a Tech Talent Database for a startup. The main branch contains the live version used by thousands of users. You want to add a new search feature. If you work directly on main and introduce a bug, you break the live system for everyone.

Instead, you create a branch called search-feature and do all your work there. main remains untouched and stable the entire time. Once the feature is complete and tested, you merge it into main.

> **The Shape of a Branch**
> *Picture main as a single stable line moving forward. A feature branch splits off from it at one point, moves forward on its own with its own commits, and then rejoins main later through a merge — at which point everything built on the branch becomes part of main too.*

Viewing Branches

```
git branch
```

The current branch is highlighted with an asterisk (*) and typically appears in green.

Creating a New Branch

```
git branch feature-name
```

Switching Between Branches

```
git switch feature-name
```

After switching, confirm with git branch again — the asterisk will have moved to the new branch.

Pushing a New Branch to GitHub

The first time you push a newly created branch, you must set the upstream:

```
git push --set-upstream origin feature-name
```

After this first push, plain git push works for every subsequent push on that branch.

A Complete Branch Workflow

```
# Make sure you're on main first
git switch main
git pull

# Create and switch to a new branch
git branch m-pesa-integration
git switch m-pesa-integration

# Do your work: create files, edit code
echo "M-Pesa API integration code here" > mpesa.txt

# Add, commit, push
git add .
git commit -m "Add M-Pesa payment integration module"
git push --set-upstream origin m-pesa-integration
```

2. Merging Branches

When your feature branch is complete and ready to be integrated into the main codebase, you merge it. The steps are:

1. Switch to the destination branch (main):

```
git switch main
```

2. Run the merge command, specifying the branch you want to merge in:

```
git merge m-pesa-integration
```

3. Push the updated main branch to GitHub:

```
git push
```

After a successful merge, main now contains everything that was in m-pesa-integration. The feature branch continues to exist separately unless you explicitly delete it.

Key Takeaways

Branches let you isolate risky, in-progress work from the stable version everyone depends on — create one with git branch, move onto it with git switch, and push it upstream the first time with --set-upstream. Once the work is done and tested, switching back to main and running git merge brings it all in. The next lesson covers what happens when two branches touch the same lines of the same file — and exactly how to resolve it when Git can't decide on its own.