SCRIPT MODULE 4

**Part 1: Why Do We Use Branches?**  
“In real-world projects, developers often work on new features or bug fixes *without disturbing the main project*. This is where branching helps — think of it like creating a parallel universe where you can test your changes safely.”

**Real-life analogy:**  
Imagine a chef experimenting with a new recipe in a separate kitchen, so the main restaurant doesn’t get affected.

**Hands-on Step-by-Step: Branching and Merging in VS Code + Git**

**🔹 Step 1: Create or Open a Git Project Folder**

bash

mkdir git-branch-demo

cd git-branch-demo

git init

📄 Create a file:

bash

echo "# My Project" > readme.md

git add .

git commit -m "Initial commit with README"

**🔹 Step 2: Create a New Feature Branch**

bash

git checkout -b feature1

“We just created a new branch named *feature1* using git checkout -b. This is a copy of the current branch where we can make changes without touching the main branch.”

**🔹 Step 3: Make Changes in Feature Branch**

Add to readme.md:

## Feature 1 section added.

Then:

bash

git add readme.md

git commit -m "Added feature 1 section"

“We committed the changes to our feature branch. Think of this as submitting your new recipe idea for review — still not affecting the original.”

**First, What's a Merge?**

**When you merge, you're combining changes from one branch (like a side road) into another branch (like the main road).**

**🔹 Step 4: Switch Back to main Branch**

bash

git checkout main

Then:

git merge feature1

You’re writing a solo document. No one else touched the file. When you're done, you just move your “Final version” label to the last line. No need for glue or edits.

“We are now merging the feature branch into the main project. If Git can auto-merge — that's called a *fast-forward* merge.”

A **Fast-Forward** happens when:

* The main branch **hasn’t changed** since you branched off
* So Git simply moves the main branch pointer forward to where your branch is

**Fast-Forward vs Recursive Merge**

**Fast-forward:** When main branch hasn’t moved ahead; Git moves the pointer forward.  
**Recursive Merge:** If both branches moved independently, Git creates a new merge commit.

**Recursive Merge (Non-Fast-Forward or 3-way Merge)**

**Layman Analogy:**

Now imagine you and your friend **both took different side roads** from the same spot, made changes, and then came back. Now you need to **combine both routes into one road**.

A **Recursive Merge** (also called a 3-way merge) happens when:

* Both branches have **new changes**
* Git must create a **merge commit** to combine the changes

"Think of fast-forward like stepping forward on the same road. Recursive is like merging two diverging roads into one new road."

**Real-Life Example:**

You wrote page 2 of a document, and your friend added page 3 at the same time. Now you need to sit down and **merge** both versions into one — and maybe talk it out if you both edited the same sentence (conflict).

**Step 5: Simulate Merge Conflict**

Go to main and make a change:

bash

git checkout main

echo "## Welcome Section" >> readme.md

git add .

git commit -m "Added Welcome Section"

Go to feature1 and make a conflicting change:

bash

git checkout feature1

echo "## Feature Introduction" >> readme.md

git add .

git commit -m "Feature intro added"

Now try merging:

bash

git checkout main

git merge feature1

You’ll get a **merge conflict**!

**Merge Conflicts: Why They Happen?**

"Merge conflicts happen when Git sees that *both branches changed the same part of a file*. Since it can’t decide whose change to keep, it asks *you* to resolve it."

**Step 6: Resolve Merge Conflicts (Manually + VS Code)**

Open VS Code:

bash

code .

Go to readme.md and you’ll see something like:

markdown

<<<<<<< HEAD

## Welcome Section

=======

## Feature Introduction

>>>>>>> feature1

➡ Choose which version you want to keep. Or combine both.  
Then:

bash

git add readme.md

git commit -m "Resolved merge conflict between main and feature1"

“In VS Code, you can use buttons like *Accept Current Change*, *Accept Incoming Change*, or *Accept Both* to quickly resolve conflicts.”

**✅ Final Result**

You’ve:

* Created a branch
* Made changes
* Merged back
* Resolved a conflict!

**Visualize in Git Graph (optional)**

Install Git Graph Extension in VS Code:

* Open Command Palette → Git Graph
* See all commits, branches, merges

“This visual helps us to see how branches were created and merged — just like a family tree.”