ENSF 381
Full Stack Web Development

Lecture 31: Git

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

UNIVERSITY OF CALGARY

Outline

- Overview of Git.
- Key features.
- Basic concepts.
- Discussion

What is Git?

- Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency.
- It was created by Linus Torvalds in 2005 for the development of the Linux kernel.
- Enable multiple developers to work on the same project simultaneously.

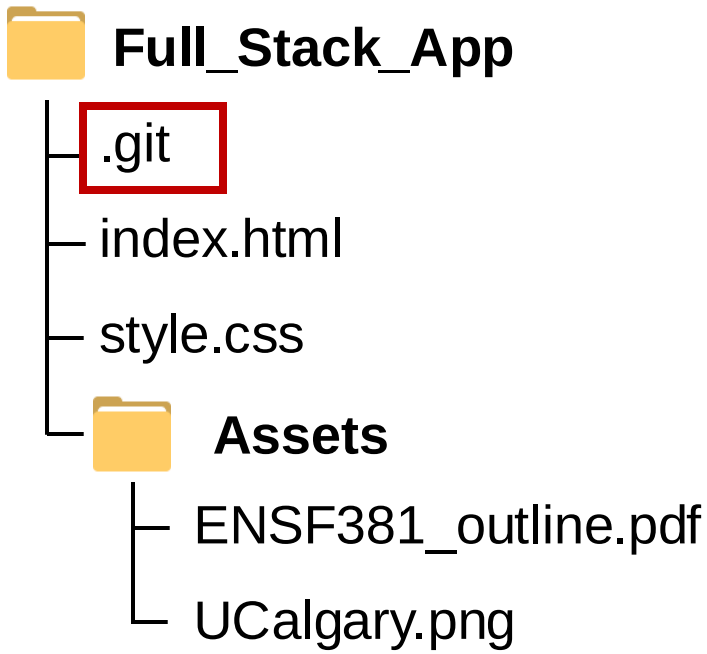
Key features of Git

- **Distributed system**: each developer can work on their local copy of the project, make changes, and then synchronize those changes with others.
- **Branching and merging**: makes it easy to create branches for separate lines of development. Changes can be made in isolation, tested, and then merged back into the main project.
- **Commit history**: maintains a detailed record of every change made to the project, providing a comprehensive history. Each change is recorded as a commit.
- **Fast and efficient**: designed to be fast and efficient, making it suitable for both small projects and large-scale, complex software development.
- **Compatibility**: compatible with various operating systems, including Windows, macOS, and Linux.

Basic concepts

- **Repository (Repo)**: is a directory or storage space where your projects can live. It can be local (on your computer) or remote (on a server).

Repository



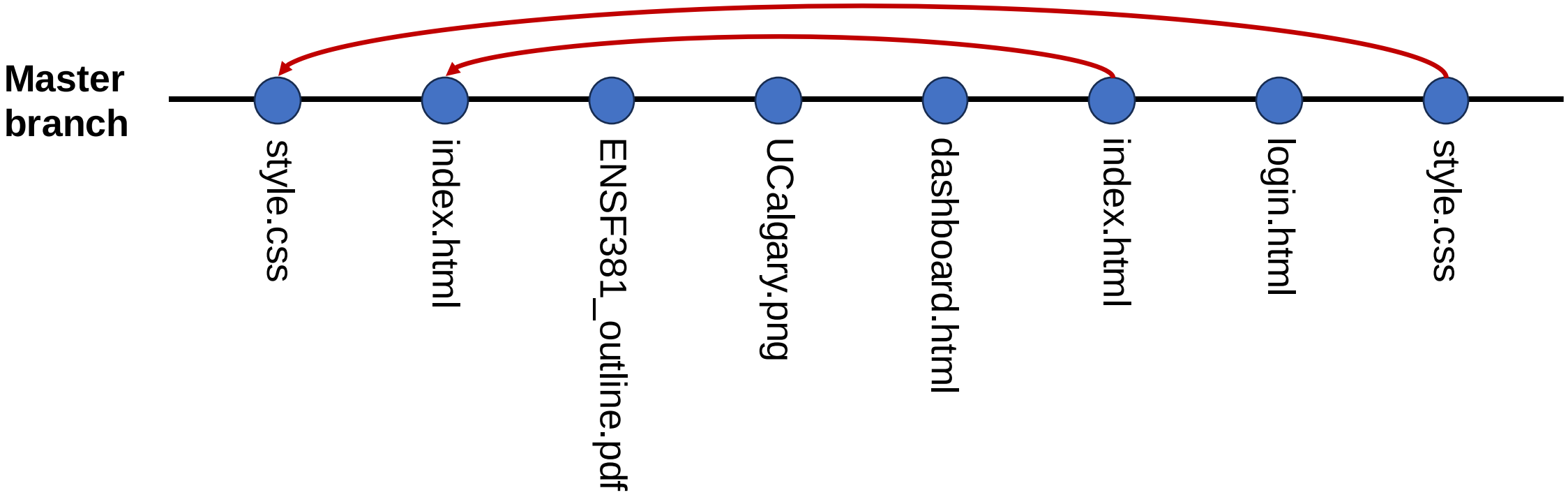
.git is a crucial component/file that stores the entire repository, including its history, configuration settings, and other important data. It is responsible for tracking and managing changes in your Git repository.

Basic concepts

- **Repository (Repo)**: is a directory or storage space where your projects can live. It can be local (on your computer) or remote (on a server).

- **Commit**: is a snapshot of changes at a specific point in time. It includes modifications, additions, or deletions of files.

Commits


Master branch
style.css
index.html
ENSF381_outline.pdf
UCalgary.png
dashboard.html
index.html
login.html
style.css

Commit - Stages

- **Working Directory (Modified)**: the current state of the project on your local machine.
- **Staging Area (Index)**: a place to stage changes before committing them to the repository.
- **Repository (Committed)**: the .git directory that stores the committed changes and project history.

Basic concepts

- **Repository (Repo)**: is a directory or storage space where your projects can live. It can be local (on your computer) or remote (on a server).

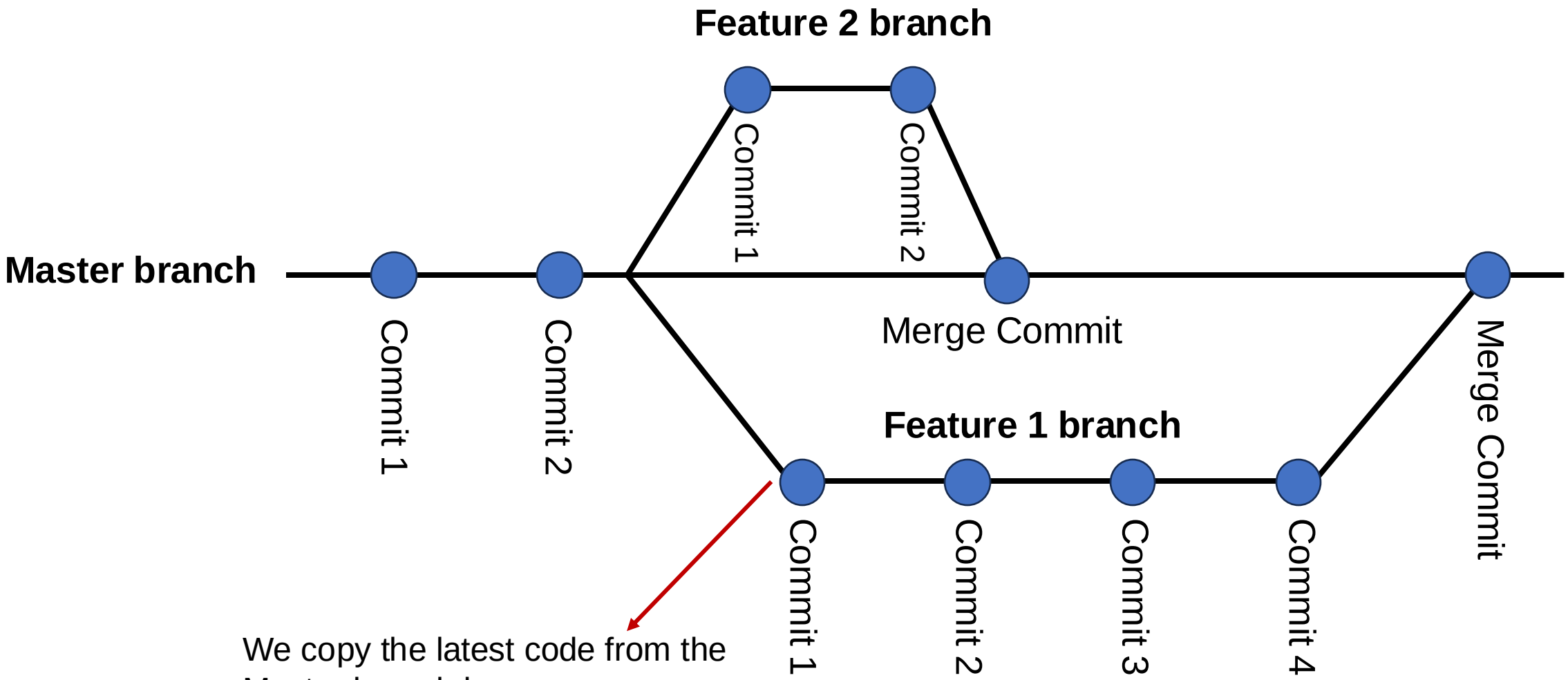
- **Commit**: is a snapshot of changes at a specific point in time. It includes modifications, additions, or deletions of files.

- **Branch**: is an independent line of development. It allows for the creation of isolated environments to work on specific features or fixes without affecting the main project.

Branches

- Allows for the creation of independent lines of development within a Git repository.
- It allows for the creation of isolated environments to work on specific features or fixes without affecting the main project.
- Branching enables parallel development and experimentation.

Branches


Feature 2 branch
Commit 1
Commit 2
Master branch
Commit 1
Commit 2
Merge Commit
Merge Commit
Feature 1 branch
Commit 1
Commit 2
Commit 3
Commit 4
We copy the latest code from the Master branch here, so we can start coding with.

Branching strategies

- **Feature branching**: each feature or task gets its own branch.
- **Release branching**: separate branches for preparing, testing, and deploying releases.
- **Hotfix branching**: immediate branches for fixing critical issues in production.

Basic concepts

- **Repository (Repo)**: is a directory or storage space where your projects can live. It can be local (on your computer) or remote (on a server).

- **Commit**: is a snapshot of changes at a specific point in time. It includes modifications, additions, or deletions of files.

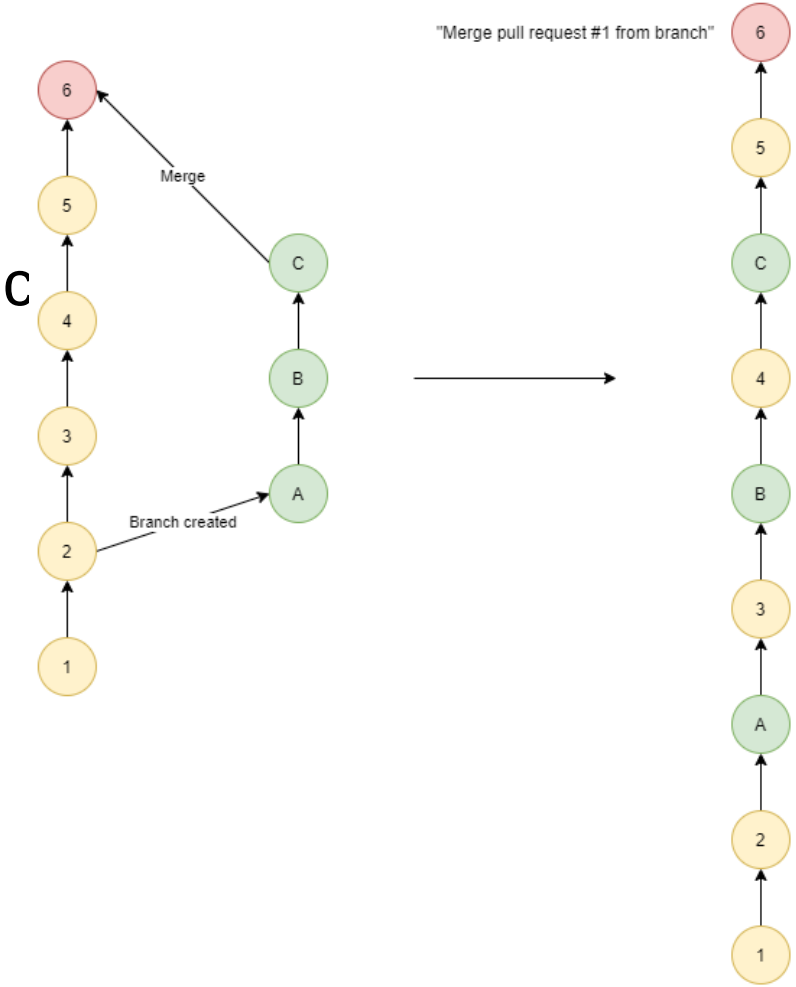
- **Branch**: is an independent line of development. It allows for the creation of isolated environments to work on specific features or fixes without affecting the main project.

- **Merge**: combines changes from different branches into a single branch, often the main branch.

Different types of merges

There are several ways to merge changes from one branch into another:

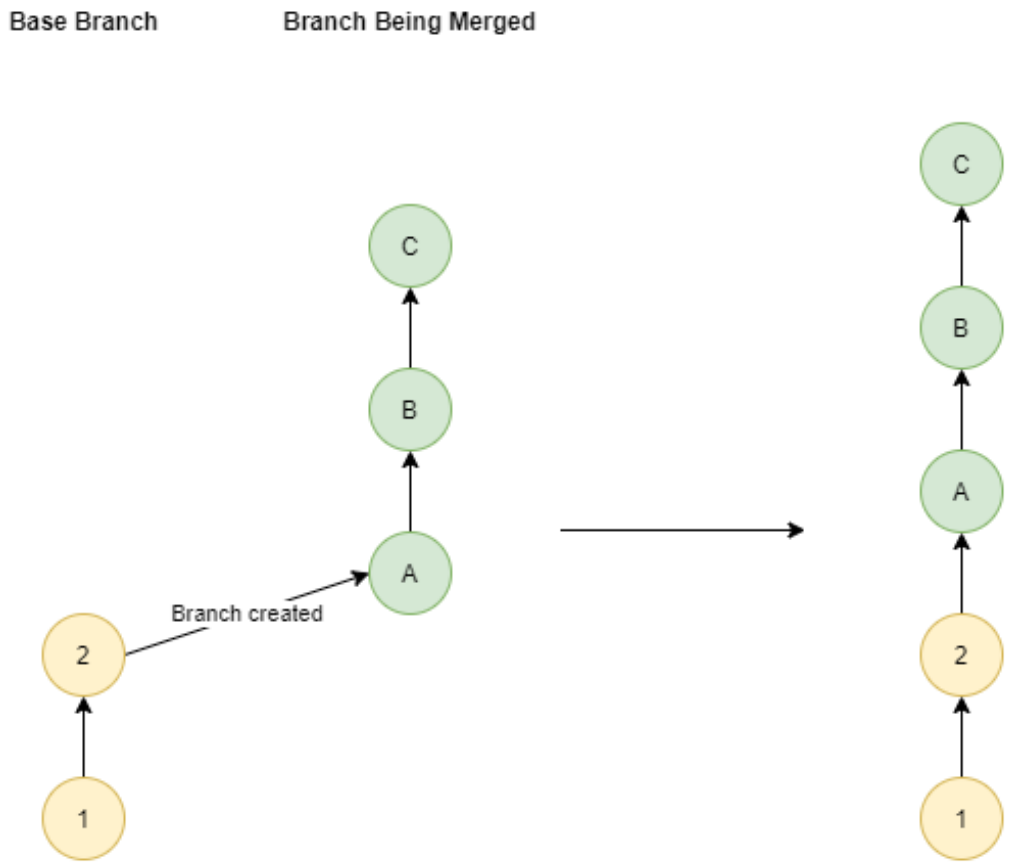
- Merge: This is the default merge strategy in Git. It creates a new commit that combines the changes from two branches. If changes in both branches do not conflic Git automatically performs the merge.



https://lukemerrett.com/different-merge-types-in-git/

Different types of merges

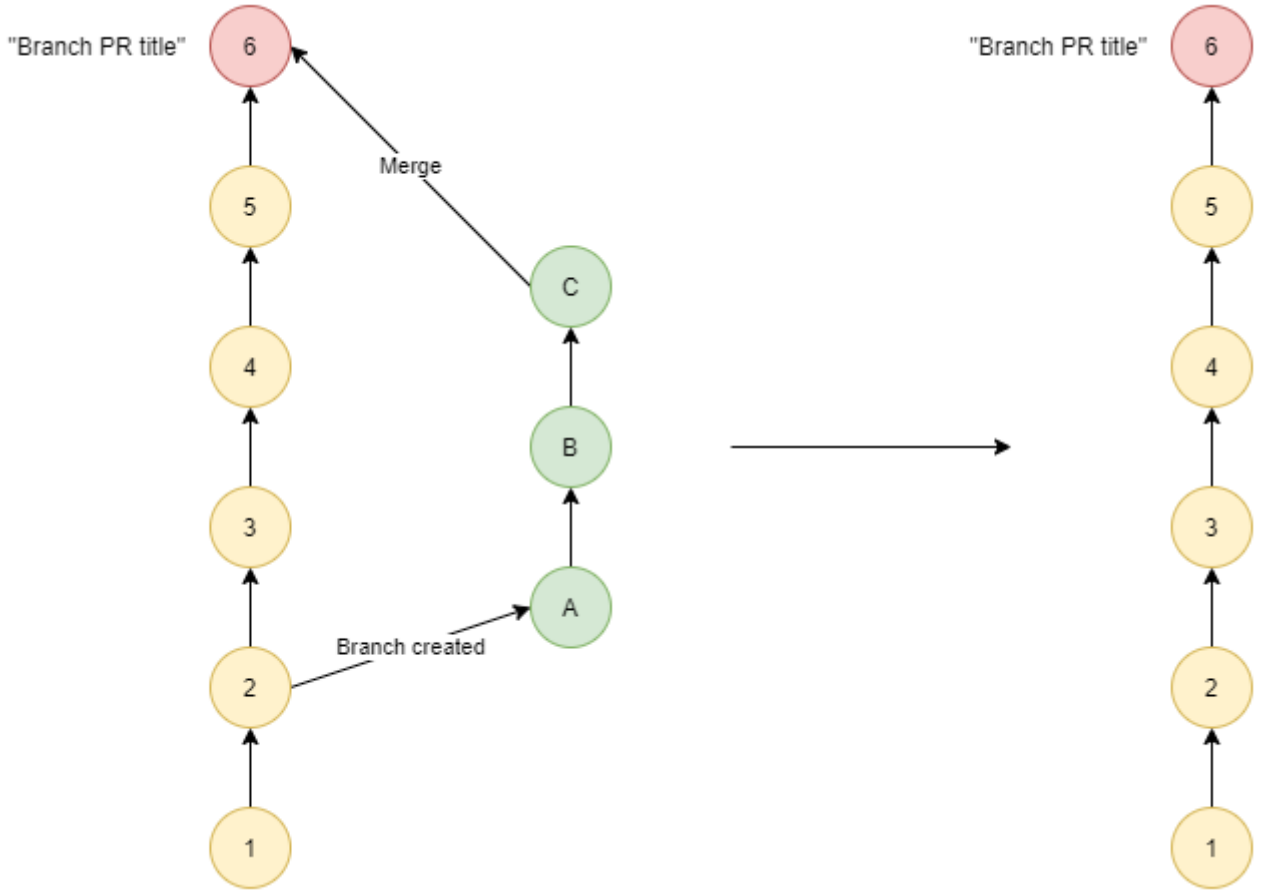
Fast-Forward Merge: occurs when the branch being merged into has not diverged since the branch was created. The branch pointer is simply moved forward to include the changes from the source branch.



https://lukemerrett.com/different-merge-types-in-git/

Different types of merges

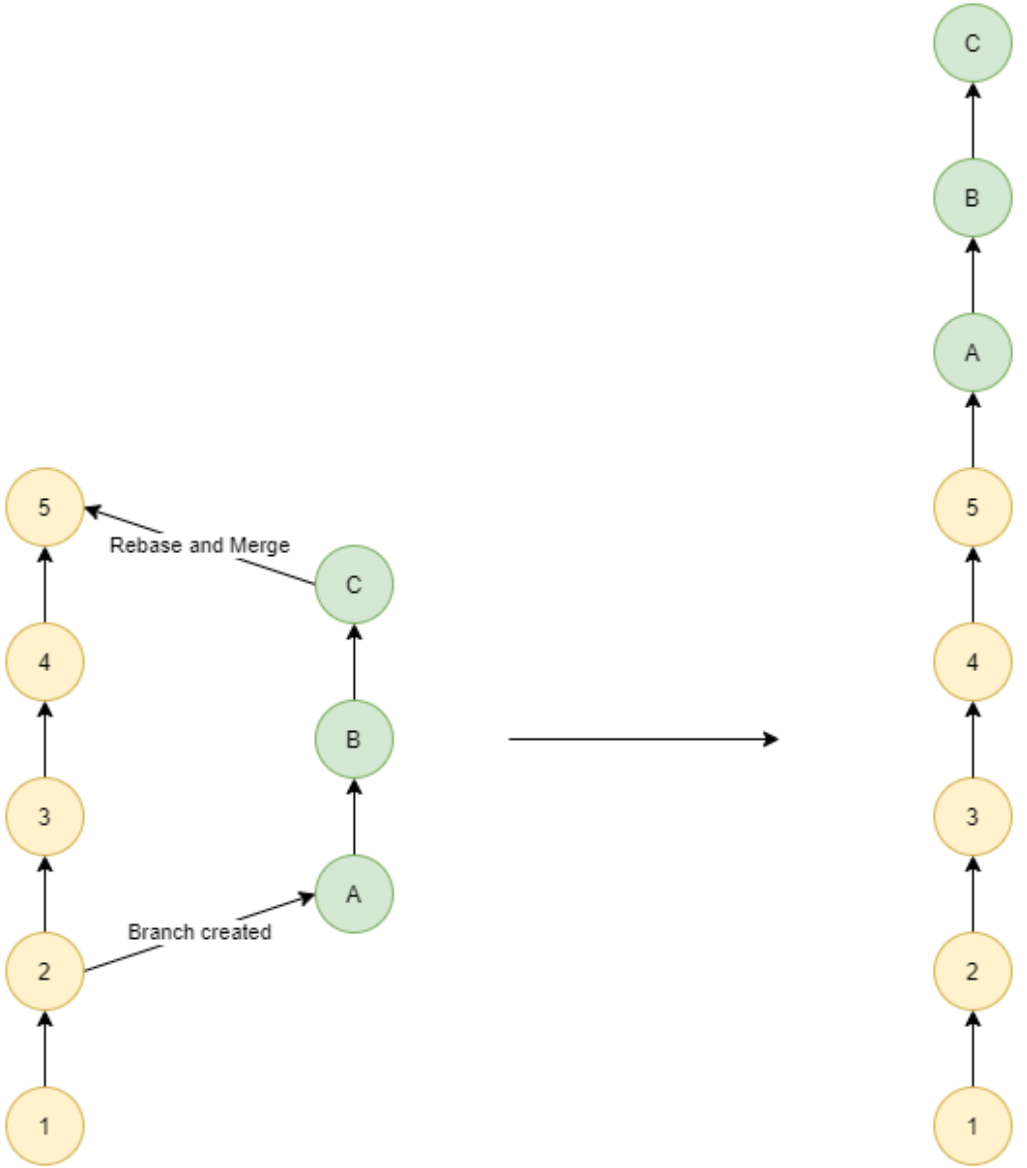
Squash Merge: Squashing combines all the changes from a feature branch into a single commit before merging. This results in a clean, linear history on the main branch.



https://lukemerrett.com/different-merge-types-in-git/

Different types of merges

Rebase and Merge: Rebasing involves moving or combining a sequence of commits to a new base commit.



https://lukemerrett.com/different-merge-types-in-git/

Shall we **separate the frontend and backend** into distinct repositories, or is it more convenient to **house them** within the same repository?

Advantages of distinct repositories

- Clear separation allows for modular development. Frontend and backend can evolve independently, making it easier to maintain and update each component.
- Different technology stacks for frontend and backend can be easily accommodated without cluttering a single repository.
- If different teams or individuals are responsible for frontend and backend, separate repositories offer more autonomy.
- Easier scalability as you can scale frontend and backend independently based on requirements.

Advantages of same repository

- A single repository provides a clear, unified history of changes, making it easier to track the evolution of the entire application.
- Versioning is simplified, and it is easier to ensure that frontend and backend versions are compatible.
- Changes that span both frontend and backend can be managed in a more coordinated manner.
- Developers can clone a single repository and have everything they need to work on the project.

Things to consider…

- **Project Size and Complexity:**
  - For small to medium-sized projects with a single team, a single repository might be more convenient.
  - For larger or more complex projects with separate teams or technologies, consider separate repositories.

- **Team Structure:**
  - If the same team works on both frontend and backend, a single repository might be more streamlined.
  - If different teams are responsible for frontend and backend, separate repositories might provide clearer ownership.

- **Development Workflow:**
  - Consider the development workflow and how changes are typically coordinated between frontend and backend.

- **Technology Stack:**
  - If frontend and backend share similar or tightly integrated technology stacks, a single repository could be more suitable.

Version Control beyond code

- Just as code evolves, data undergoes changes. Versioning data enables tracking changes, ensuring reproducibility, and facilitating collaboration among data scientists.
- Tools like DVC (Data Version Control) offers features tailored for versioning large datasets, managing metadata, and enabling data lineage.
- Versioning ML models allows tracking model changes, comparing performance across iterations, and rolling back to previous versions if necessary.
- Frameworks like MLflow provide comprehensive solutions for versioning models, managing experiments, and deploying models at scale.

Questions

Reminder: The deadline for Assignment 4 is **today at 11:59 PM**.