GIT

Install GIT

Content

GIT

What is VCS ?

Introduction

- Version Control Systems (VCS) are tools that help manage changes to source code over time.
- VCS tracks every modification to the code in a special kind of database.
- With VCS, developers can compare and revert changes to fix mistakes.

GIT

Introduction

VCS Types

1. Centralized Version Control Systems (CVCS):
   - Uses a single server to store all versions of the project.
   - Example: Subversion (SVN).
2. Distributed Version Control Systems (DVCS):
   - Every contributor has a local copy of the entire project history.
   - Example: **Git**, Mercurial.

GIT

Introduction

What is Git?

- A distributed version control system.
- Tracks changes in source code during software development.
- Facilitates collaboration among developers.

GIT

Introduction

Why use Git?

- Efficiently manages versions of code.
- Enhances collaboration and workflow.
- Provides powerful branching and merging capabilities.

GIT

Introduction

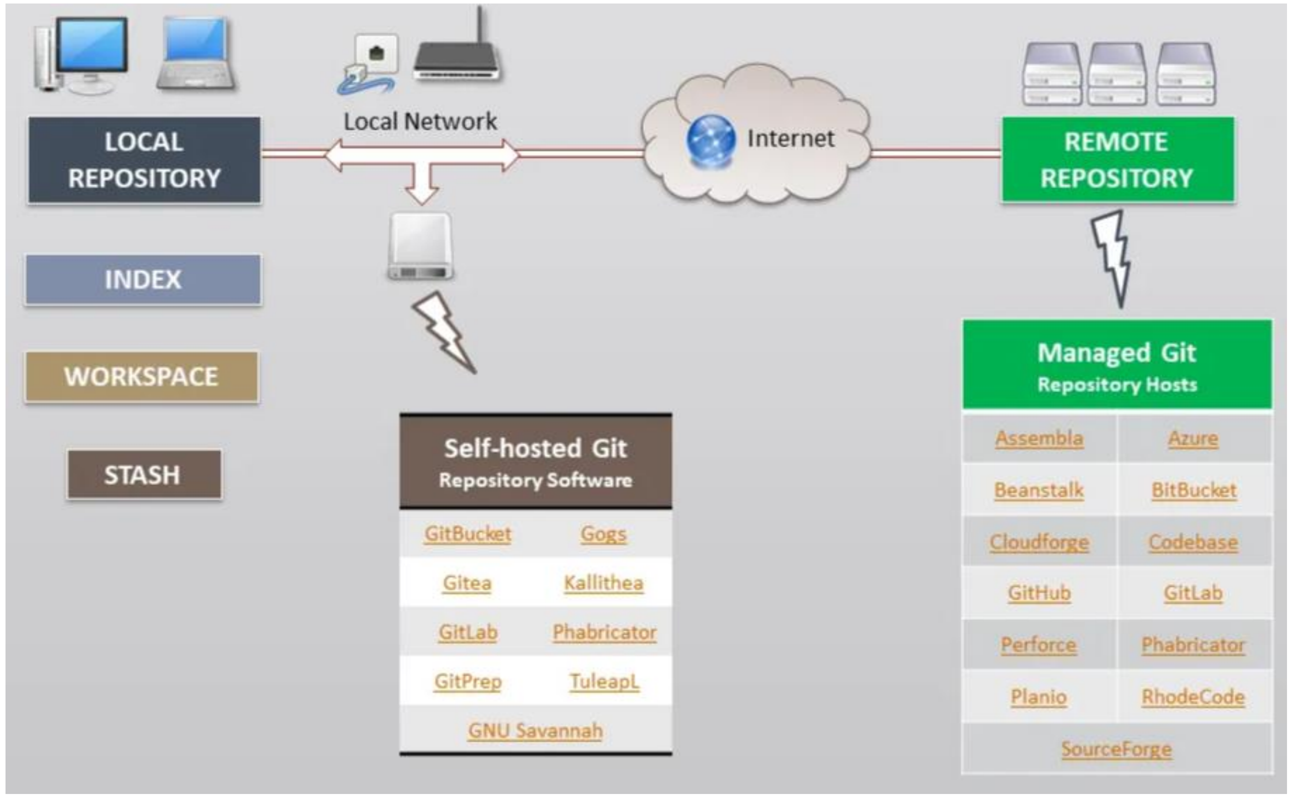
Overview of Key Concepts

- **Repository**: A storage for all the code and its history.
- **Commit**: A snapshot of the project at a point in time.
- **Branch**: A parallel version of the repository.
- **Merge**: Integrates changes from different branches.

GIT

Version Control with Git

Repositories



GIT

Types of Repositories

Repositories

Remote Repository:

- Stored on a server, which is accessible over the internet.
- Allows collaboration with other developers.
- Use case: Sharing your project with team members, continuous integration, and deployment.

GIT

Types of Repositories

Repositories

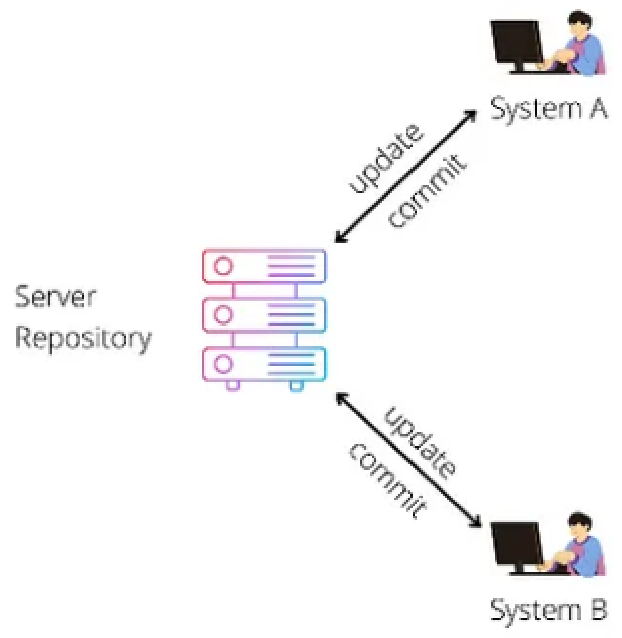
Local Repository:

- Stored on your local machine.
- Allows you to work on your project without needing an internet connection.
- Use case: Making and testing changes privately before sharing with others.

GIT

Centralized Version Control System

Repositories



GIT

Repositories

Centralized Version Control System

Server and Client Model:

- Server is the master repository containing all versions of the code.
- Clients can access, pull, and push files to/from the server.

Basic Workflow:

- Get the latest code from the central repository.
- Make changes locally.
- Commit or merge changes back into the main repository.

Advantages:

- Enables easy collaboration with multiple developers.
- Centralized location for all code versions.

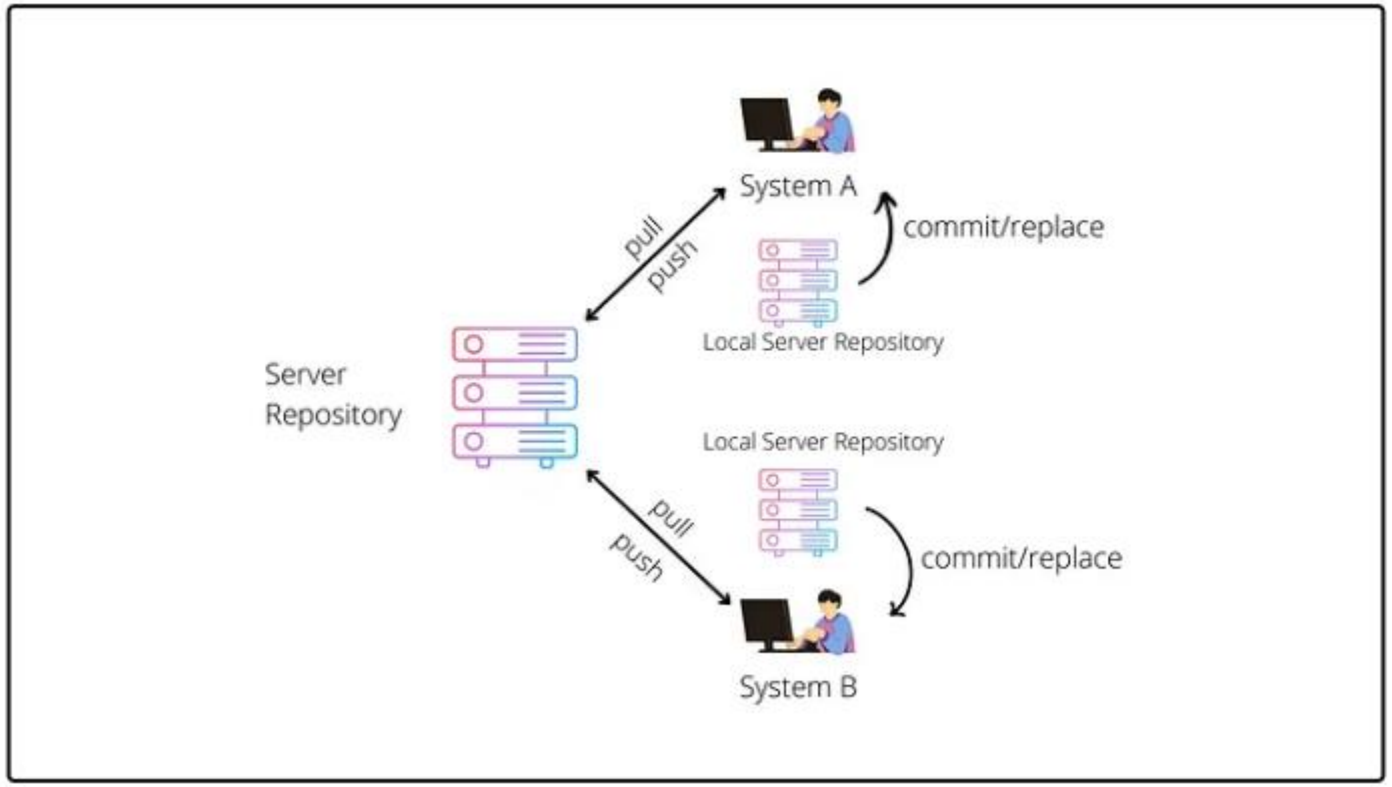
Disadvantages:

- Single point of failure: versioning and collaboration halt.

GIT

Distributed Version Control System

Repositories



GIT

Repositories

Distributed Version Control System

Server-Client Model:

- Each developer has their own local repository.
- Full repository history and branches are mirrored locally.

Basic Workflow:

- Developers clone the entire repository.
- Work locally, commit changes to local repository.
- Push changes to share with others, pull to update local repo.

Advantages:

- Each local repo serves as a backup.
- Easier branching, merging, and parallel development.

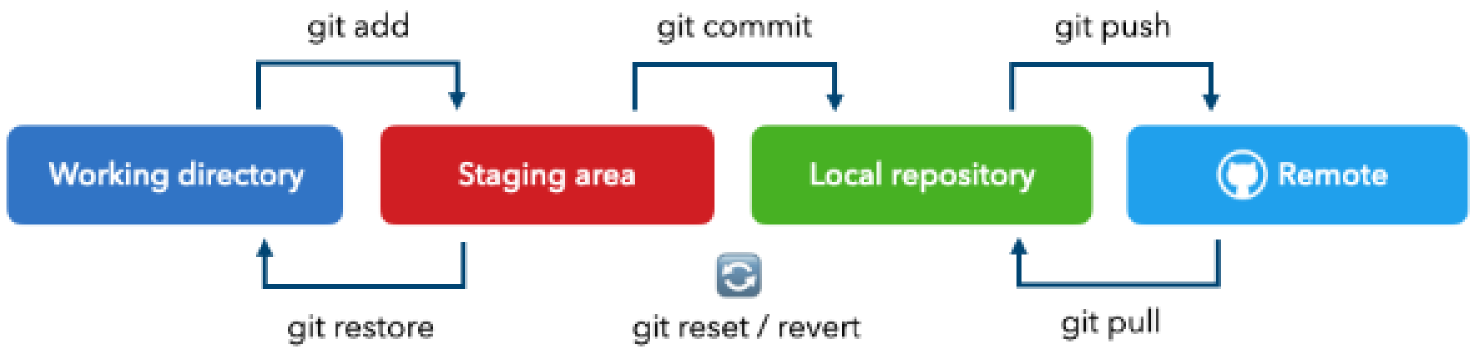
Disadvantages:

- Managing distributed repos can be more complex.
- Potential for conflicts when syncing changes.

GIT

Version Control lifecycle

Repositories



GIT

Repositories

The Working Directory

Definition:

Is the folder on your local machine where you work on files.
It contains the **Local Repository**

Purpose:

Allows you to make changes, edit, add, or remove files before staging and committing them.

Commands:

- git status: Shows the status of the working directory.
- git diff : Shows the state of the working directory and staging area.
- git checkout -- <file>: revert to last committed state).
- git clean -f: Remove untracked files from the working directory.

GIT

Repositories

Staging Area

Definition:

A temporary area where changes are prepared before committing.
Acts as an intermediate between the working directory and the local repository.

Purpose:

Allows selective inclusion of changes.
Facilitates better commit management.

Commands:

- git status: Shows the state of staging area.
- git add <file>: Adds changes to the staging area.
- git reset <file>: Unstages the changes for a specific file.

GIT

Repositories

Local Repository

Definition:
A repository on your local machine.
Stores the complete history of commits.

Purpose:
Allows offline access to the project's history.
Facilitates local branching and merging.

Commands:

- git commit -m "message": Records changes to the local repository.
- git log: Shows the commit history.
- git reset --soft <commit>: Resets to a specific commit but keeps changes in the staging area.
- git reset --hard <commit>: Resets to a specific commit and discards all changes.

GIT

Repositories

The Origin (Remote Repository)

Definition:

A remote repository typically hosted on a server.
Centralized repository for collaboration.

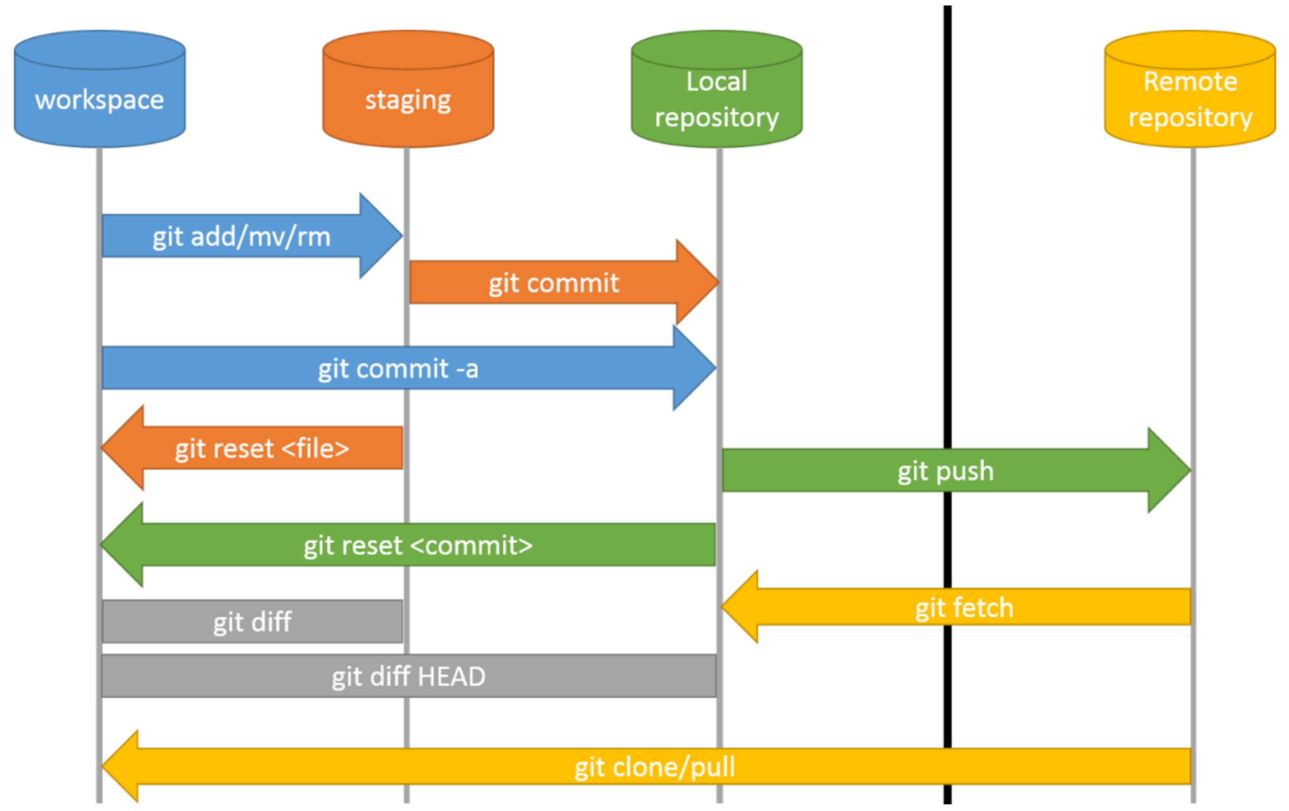
Purpose:
Facilitates sharing and collaboration.
Acts as the primary source of truth for the project.

Commands:
- git remote add origin <url>: Adds a remote repository.
- git push origin <branch>: Pushes local changes to the remote repository.
- git pull origin <branch>: Fetches and merges changes from the remote repository.

GIT

Actions

Repositories



GIT

Repositories

Create New Repository

- Create New local repository.
- Add relevant files & folders to the local repository.
- Commit the changes with a message.
- Link the local repository to a remote repository.
- Push the commit of your changes to master/main Branch on the remote repository.

GIT

Repositories

Maintain Existing Repository

- Clone the remote repository.
- Navigate to the Cloned Repository.
- Do the relevant changes .
- Commit the changes to stage.
- Push the commit to master on the remote repository.

GIT

Update Existing Local Repository from Remote Repository

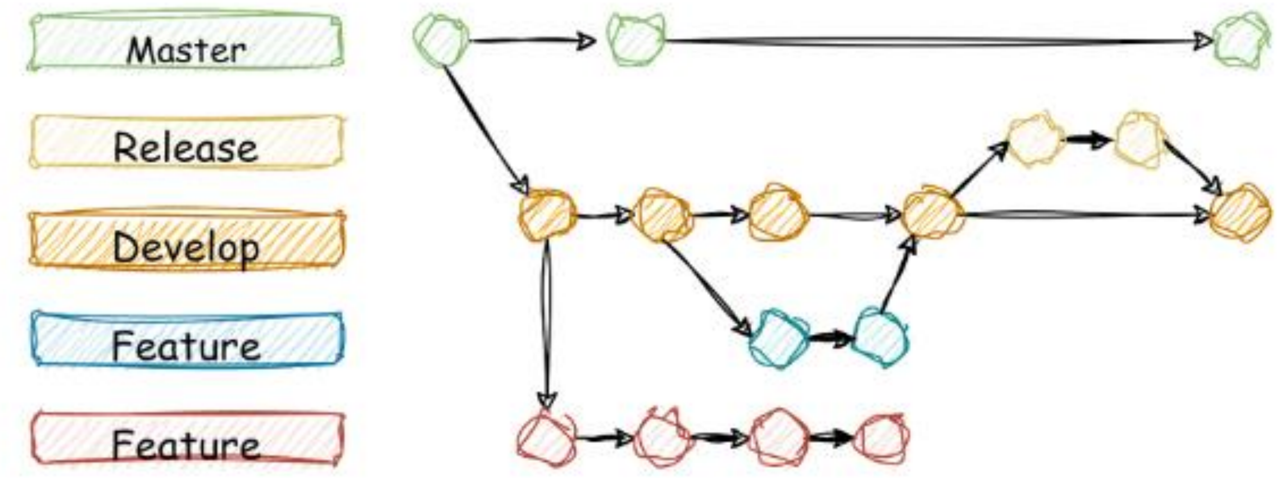
Repositories

- Fetch Changes from Remote Repository.
- Merge Changes into Local Branch.
- Pull Changes from Remote Repository.

GIT

Branches

Branches



GIT

Branches

What is a Branch?

- A branch is a parallel version of a repository.
- Branch can exist in both local and remote repositories.
- Branch allows you to work on different features or fixes independently.
- The **master** / **main** branch is the default branch.

GIT

Branches

Why are we use Branch?

- Allows multiple work on different features simultaneously.
- Each branch represents an isolated environment.
- Branches enable a structured workflow, such as feature branching or bug fixing.
- Preserves a clear history of changes and developments.

GIT

Branches

Types of branches in Git

Master Branch:

- Default branch in a Git repository.
- Represents the main line of development and stable version of the code.

Feature Branches:

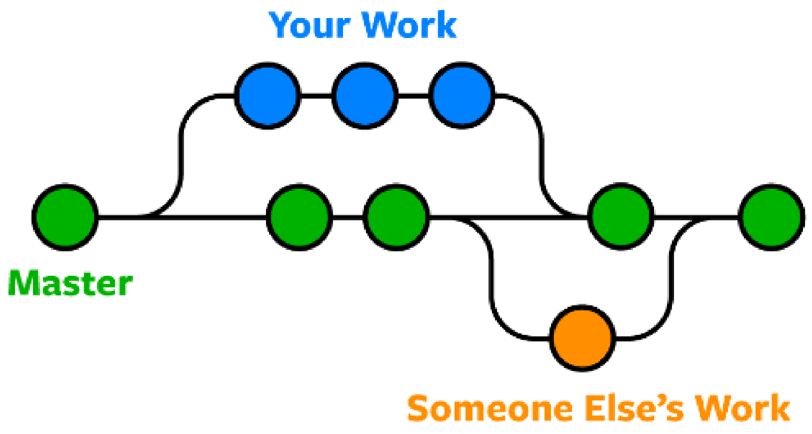
- Created for developing new features or improvements.
- Separate from the main branch.
- Merged back into the main branch when the feature is complete.

Bugfix Branches:

- Created for fixing specific bugs in the code.
- Separate from the main branch.
- Merged back into the main branch when the bug is fixed.

GIT

Branches



- Create a new branch: git branch branch_name
- Switch to existing branch: git checkout branch_name
- Create a new branch and switch to it : git checkout -b branch_name
- List local branches : git branch
- Delete the specified branch : git branch -d branch_name
- Delete in force the specified branch : git branch -D branch_name
- Rename the branch name : git branch -m new_name
- Merge the specified branch into the current branch:
  git merge specified_name
- Rebases the current branch onto the specified branch:
  git rebase specified_name
- Pulls changes from the remote repository for the specified branch :
  git pull origin main

GIT

Branches

What is Git Merge?

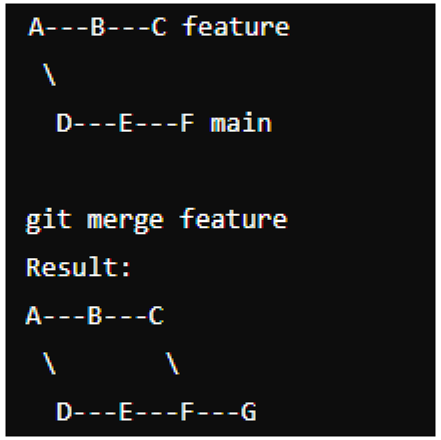
Definition:

- Combining multiple branches into one.

How it Works:

- Creates a new commit that includes changes from both branches.
- Keeps the history of both branches intact.

Visual Example:



```
A---B---C feature
 \
  D---E---F main


git merge feature
Result:
A---B---C
 \       \
  D---E---F---G
```

GIT

Why Use Git Merge?

Branches

- **Preserve History:** Retains the complete history of changes.
- **Collaboration:** Allows multiple team members to work on different features simultaneously.
- **Stability:** Useful for integrating changes without rewriting history.

GIT

Branches

Why Use Git Merge?

Merge branch

Switch to the Main Branch `git checkout main`

Pull the Latest Changes `git pull origin main`

Merge the Feature Branch `git merge feature-branch`

Resolve any conflicts `git add <resolved-file>`

Commit the Merge `git commit`

GIT

Branches

What is Git Rebase?

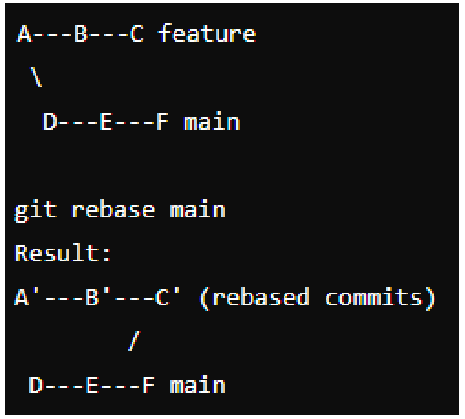
Definition:

Reapplying commits on top of another base commit.

How it Works:

- Moves or combines a sequence of commits to a new base commit.
- Changes the commit history.

Visual Example:



```
A---B---C feature
 \
  D---E---F main


git rebase main
Result:
A'---B'---C' (rebased commits)
        /
 D---E---F main
```

GIT

Why Use Git Rebase?

Branches

- **Cleaner History**: Produces a linear and cleaner commit history.
- **Simpler Log**: Easier to follow the project history.
- **Avoid Merge Commits**: Useful for avoiding unnecessary merge commits in the history.

GIT

Branches

How to Git Rebase?

Rebase branch

rebasing allows you to integrate changes from one branch into another by applying the commits individually.

1. Ensure you are on the branch to be rebased
```
git checkout feature-branch
```
2. Fetch the latest changes from the remote repository
```
git fetch origin
```
3. Rebase the branch onto the target branch
```
git rebase main
```
4. Resolve any conflicts
```
git rebase –continue
```

GIT

Branches

Merge VS. Rebase?

Git Merge:

- Retains the complete history, shows true branch structure.
- Can result in complex commit history with many merge commits.

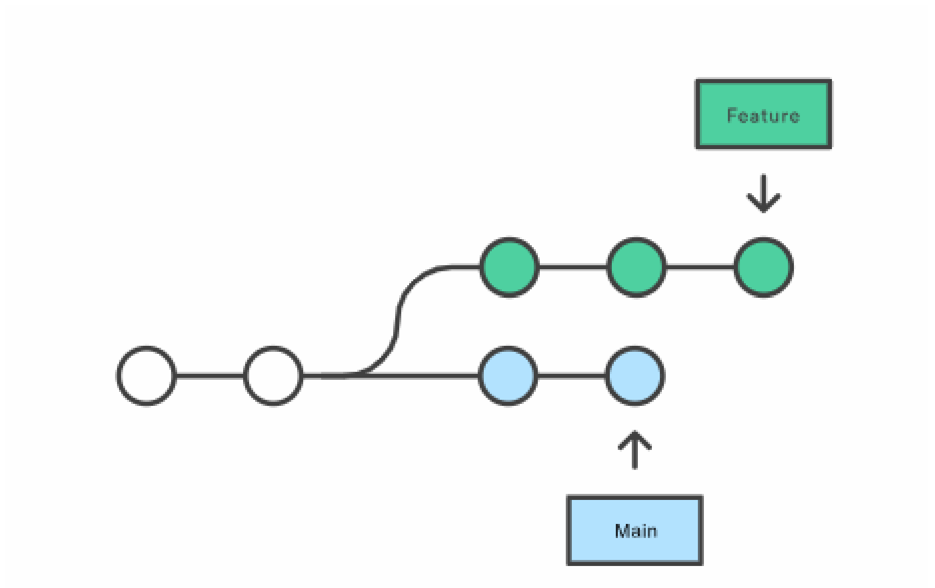
Git Rebase:

- Cleaner, linear commit history.
- Rewrites history, can be dangerous if not used carefully (especially with shared branches).

GIT

Merge VS. Rebase?

Branches



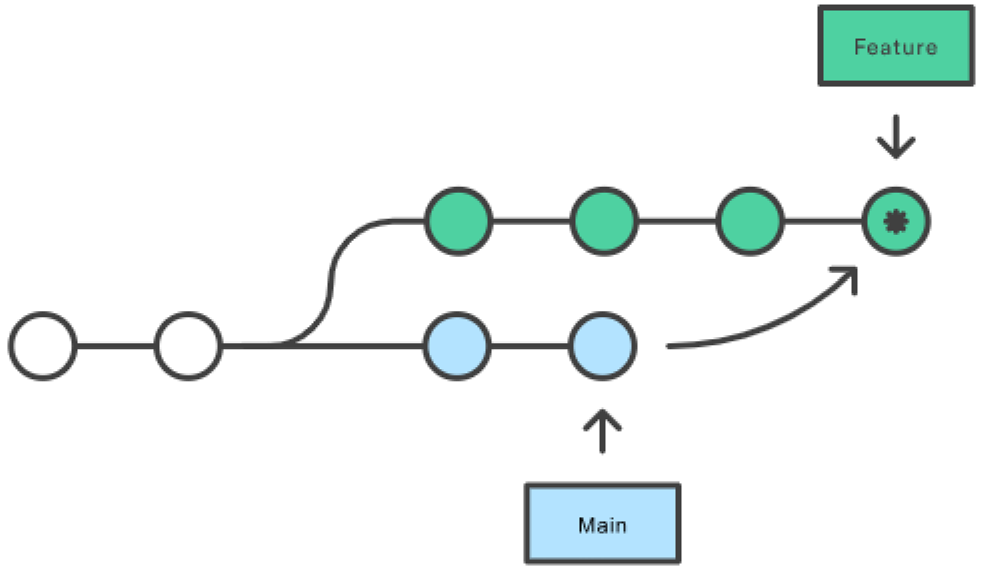
GIT

Branches

Merge

git checkout feature
git merge main

OR
git merge feature main

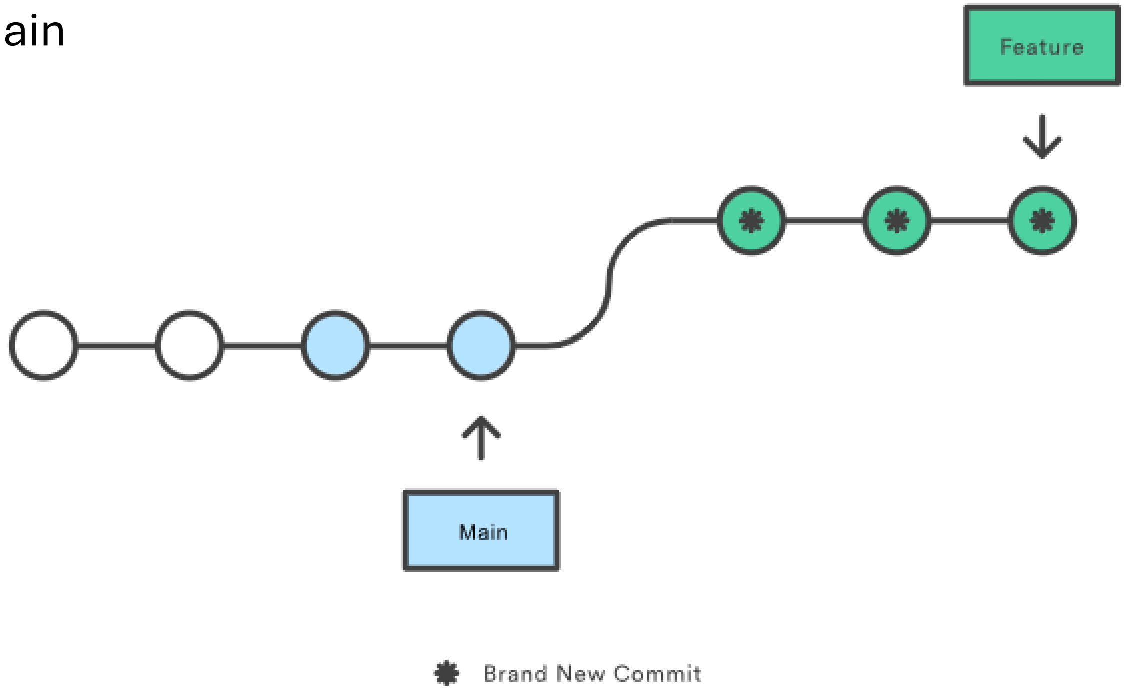


GIT

Branches

Rebase

```
git checkout feature
git rebase main
```



GIT

Conflicts

What is Conflict ?

Definition: A conflict occurs when Git is unable to automatically merge changes in a file.

Common Scenarios:

- Concurrent modifications of the same file.
- Changes in the same lines of a file.

GIT

When Do Conflicts Occur?

Conflicts

- **Merging Branches:** Conflicts occur when merging branches with changes to the same lines of code.
- **Rebasing:** Conflicts can arise during rebasing if the base branch has conflicting changes.
- **Cherry-Picking:** Conflicts may happen when cherry-picking commits that introduce overlapping changes.

GIT

Conflicts

Identifying Conflicts

Visual Indicators:
Conflicted files are marked with conflict markers
<<<<<<<, =======, >>>>>>>.

Commands to Detect Conflicts:

- git status: Shows conflicted files.
- git diff: Shows the differences between branches.

GIT

Conflicts

Example of a Conflict

Scenario:

Two branches, *feature-a* and *feature-b*, both modify **file.txt**.

Conflict Markers:

```
<<<<<<< HEAD
        Changes from the current branch
        =======
        Changes from the other branch
>>>>>>> feature-b
```

GIT

Resolving Conflict

Conflicts

Steps:

1. Open the conflicted file and locate conflict markers.
2. Edit the file to combine the changes or choose one side.
3. Remove conflict markers and save the file.
4. Add the resolved file: git add <file>.
5. Continue the merge: git commit.

GIT

Ignore

Git Ignore

- .gitignore files are used to specify which files and directories to ignore in a project
- . gitignore preventing certain files from being tracked and included in version control.
- . gitignore generally useful for preventing certain build artifacts, temporary files, and sensitive information.
- The file . Gitignore is created under the root dir of the repository

GIT

Ignore

- .Ignore all files with a specific extension *.log , *.tmp
- Ignore a specific file like secret.txt
- Ignore a directory like /node_modules /dist

Git Ignore

GIT

Stash

Git Stash

- Git stash is a powerful feature that allows you to save your uncommitted changes temporarily without committing them.
- Git stash Keeps your working directory clean.
- Git stash Allows for context switching without losing work.
- Git stash Helps in managing work in progress.

GIT

Git Stash

Stash

Workflow :

1. Stash Changes: Run **git stash** to stash your current changes.

2. Switch Branch: Checkout to a different branch

3. Apply Stash: After completing work on the other branch, return to the original branch and run **git stash apply** to restore the stashed changes.

GIT

Git Stash

Stash Commands

- **git stash**: Stashes the changes in a dirty working directory.
- **git stash list**: Lists all stashed changes.
- **git stash apply**: Applies the most recent stash.
- **git stash apply stash@{n}:** Applies a specific stash.
- **git stash drop**: Deletes a specific stash.
- **git stash pop**: Applies the most recent stash and removes it from the stash list.

GIT

What are Issues?

Issue

Issues are a way to track

- tasks
- enhancements
- bugs
- feature requests

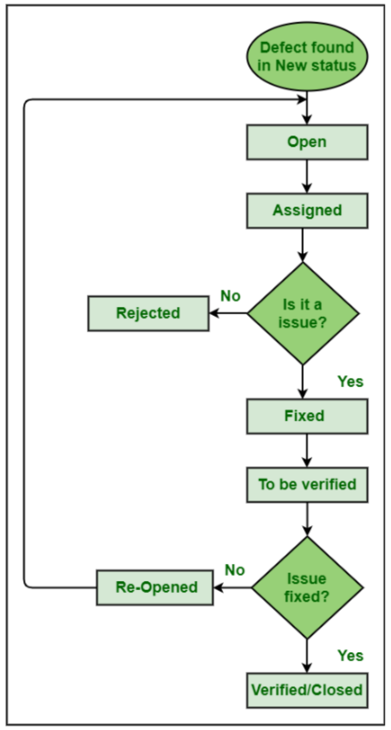
Issue is used to

- Categorize and prioritize tasks.
- Clear documentation of bugs and features.
- Monitor the status of tasks and assignments.
- Allow team members to discuss and refine tasks.

GIT

Issues Life cycle

Issue



Defect Life Cycle

GIT

Issues

What are Pull Requests (PR)?

A Pull Request is a way to propose changes to a repository.

Coming to Submit :

- New features
- Bug fixes
- Documentation changes.

PRs are used to:

- Enforce code reviews and discussions before merging.
- Facilitate team collaboration and feedback.
- Provide a history of changes and decisions.
- Ensure changes are tested and do not introduce new issues.

GIT

Pull Requests workflow

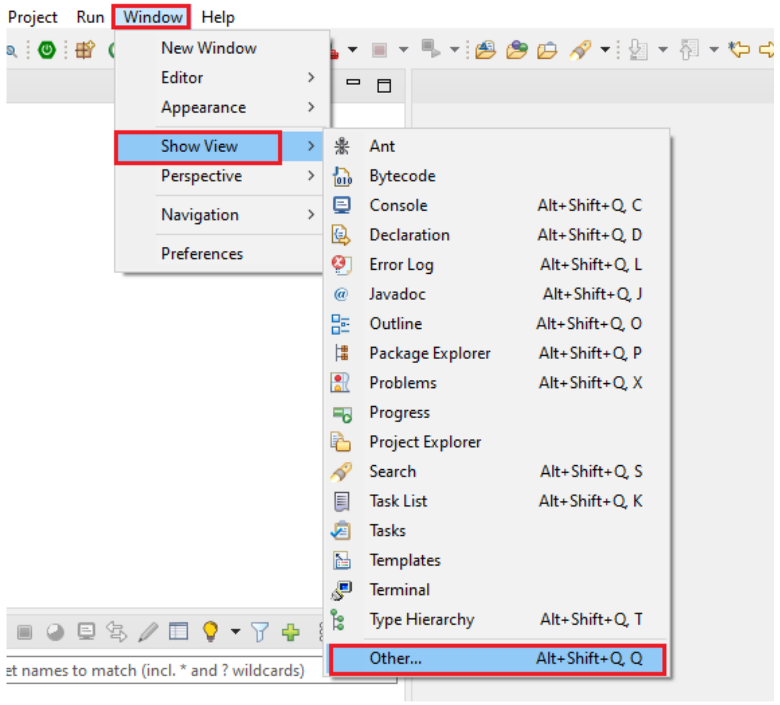
Issues

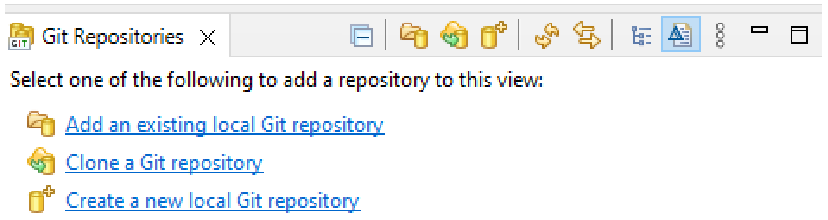
- Create a branch
- Commit changes
- Open a Pull Request
- Code review and discussion
- Merge into the main branch

GIT

Eclipse Integration

Issues





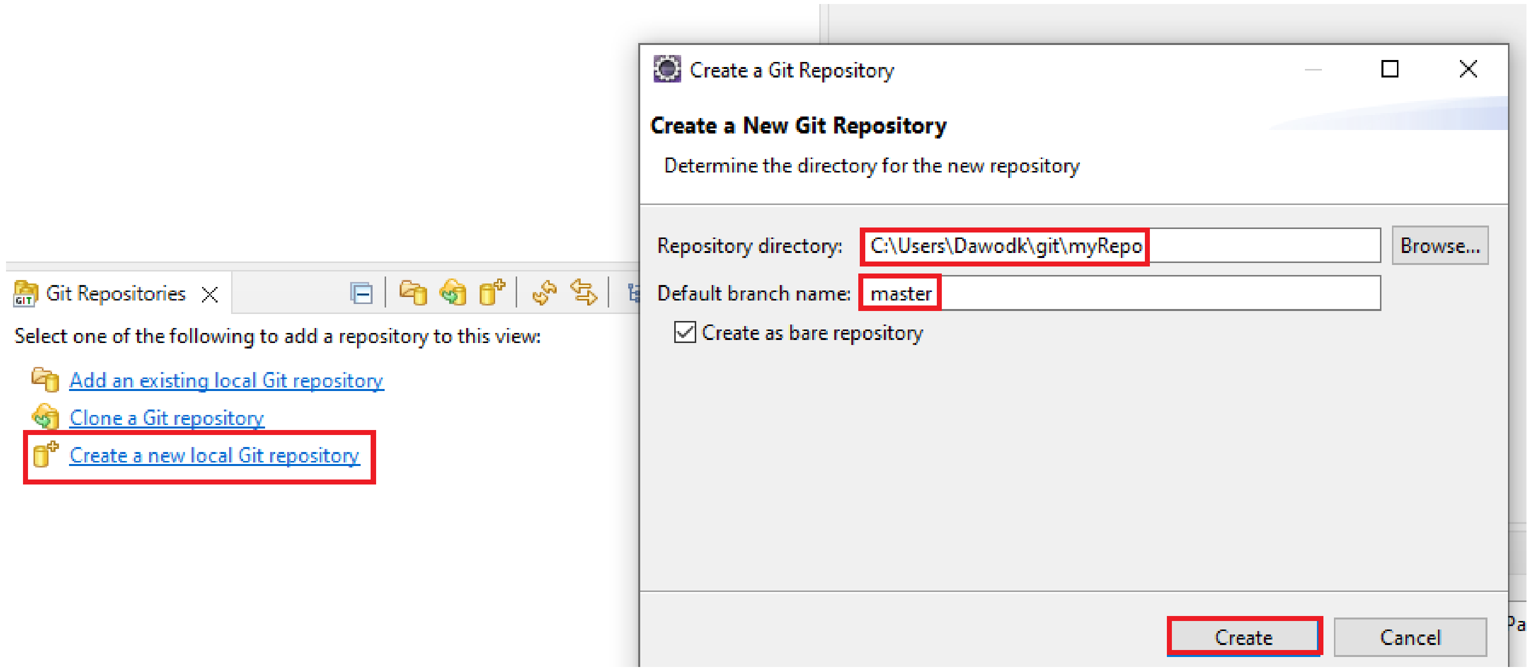
Select one of the following to add a repository to this view:

Add an existing local Git repository
Clone a Git repository
Create a new local Git repository

GIT

Eclipse Integration

Issues


Git Repositories

Select one of the following to add a repository to this view:

- Add an existing local Git repository
- Clone a Git repository
- Create a new local Git repository

Create a Git Repository

Create a New Git Repository

Determine the directory for the new repository

Repository directory: C:\Users\Dawodk\git\myRepo

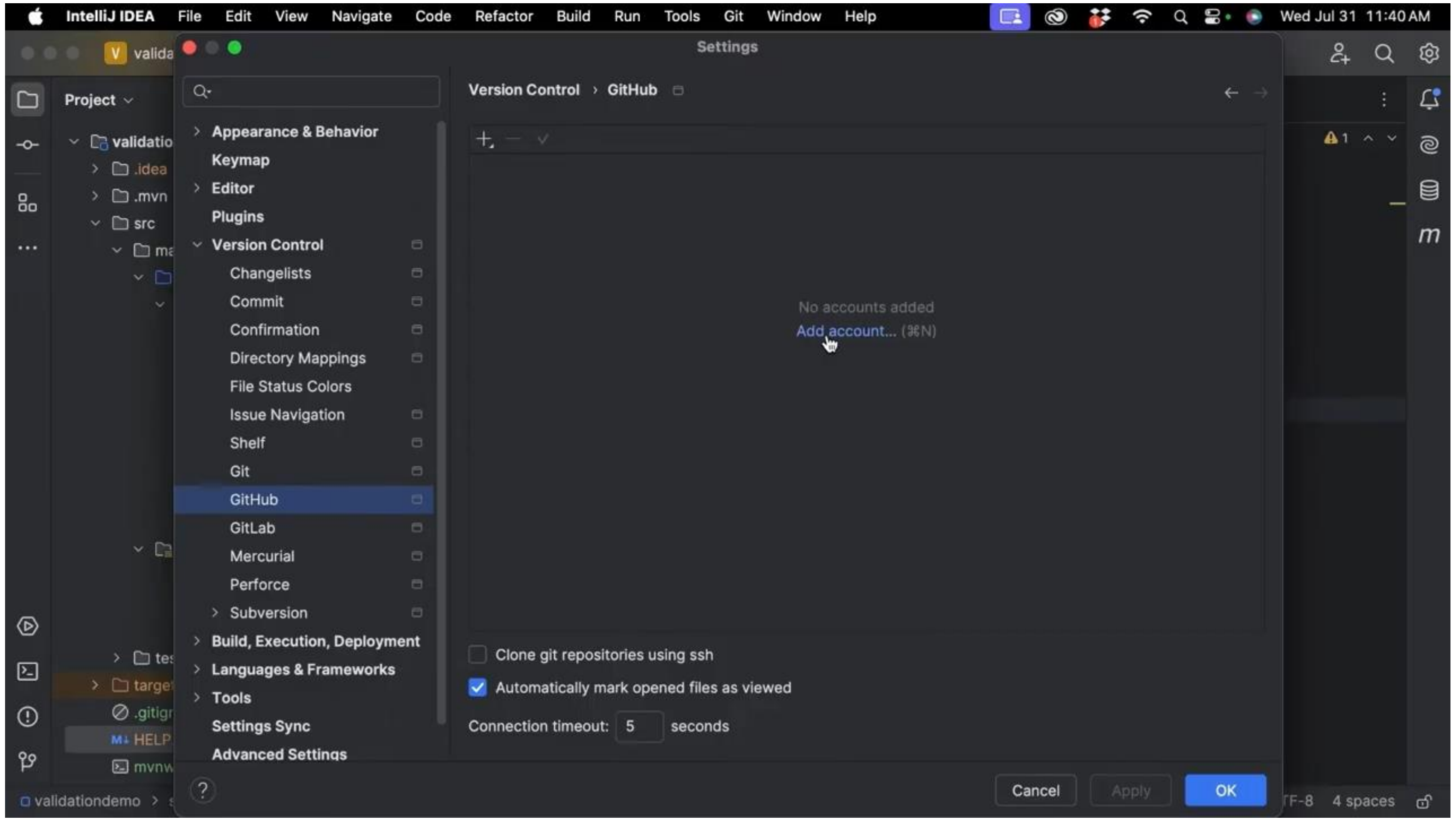
Default branch name: master

☑ Create as bare repository

GIT

Integration

IntelliJ IDEA Integration



Clone git repositories using ssh [ ]
Automatically mark opened files as viewed [✓]
Connection timeout: 5 seconds

GIT

GIT

Thank You !!