Git

What is Git?

Distributed version control system for tracking changes in source code during software development.

Key Aspects:

- Developed by Linus Torvalds.
- Allows distributed and non-linear version control.

Benefits of Using Git in Java Development:

- Collaborative Java Development:

- Enables multiple developers to collaborate on Java projects seamlessly.

- Branching for Feature Development:

- Facilitates the creation of feature branches, allowing developers to work on specific features or bug fixes independently.

- Codebase History in Java Projects:

- Maintains a detailed history of changes, providing insights into when and why changes were made in the Java codebase.

- Dependency Management with Maven/Gradle:

- Integrates smoothly with build tools like Maven or Gradle, aiding in managing Java project dependencies.

- Automated Build and Continuous Integration:

- Supports automated builds and continuous integration, ensuring code changes are thoroughly tested before integration.

- Code Review Workflow:

- Facilitates code review workflows through pull requests, enhancing code quality and adherence to standards.

- Efficient Handling of Java Libraries and Dependencies:

  - Manages external libraries and dependencies with version-controlled configuration files.

- Remote Repositories for Distribution:

  - Enables the distribution of Java code through remote repositories, fostering collaboration among geographically dispersed team members.

- Efficient Debugging and Rollback:

  - Provides tools for efficient debugging and the ability to roll back to previous commits in case of issues or bugs.

- Integration with Java IDEs:

  - Integrated support in Java IDEs (e.g., IntelliJ IDEA, Eclipse), allowing developers to perform Git operations directly within the development environment.

Git Tools & Packages

Git Tools:

- Git Command-Line Interface (CLI):

  - Core tool for executing Git commands via the terminal or command prompt.

- Git GUI Clients:

  - Graphical User Interface tools providing a visual representation of Git operations.
  - Examples: GitHub Desktop, Sourcetree, GitKraken.

- IDE Integrations:

  - Built-in Git integration in IDEs like IntelliJ IDEA, Eclipse, and Visual Studio Code.

- Git Bash:

  - Command-line terminal for Git on Windows, offering a Unix-like shell.

- Web-Based Platforms:

- GitHub, GitLab, Bitbucket - Host Git repositories and offer collaboration features.

- GitKraken:

- Popular Git GUI client with a visually intuitive interface.

Git Terminology:

- Repository:

- Data structure storing metadata for a set of files/directories, typically project files.

- Commit:

- Snapshot of changes with a unique identifier, containing info about changes, author, timestamp.

- Branch:

- Parallel line of development allowing independent work on features/bug fixes.

- Merge:

- Integrating changes from one branch into another, combining changes.

- Pull Request (PR):

- Request to merge changes from one branch into another, often used in collaboration.

- Fork:

- Personal copy of someone else's repository for experimentation without affecting the original.

- Clone:

- Creating a local copy of a remote repository for local development.

- Remote:

- Version of the repository stored on a server, e.g., GitHub, GitLab.

- Pull

- Fetching changes from a remote repository and merging them into the local branch.

- Push:

- Sending committed changes from a local repository to a remote repository.

- HEAD:

- Reference pointing to the latest commit in the current branch, representing the current state.

- Stash:

- Mechanism to temporarily save changes not ready to be committed, enabling branch switching.

Git Basic Commands

Staging and Commit:

- Initialization

```bash
git init
```

- Initializes a new Git repository in the current directory.

- Clone

```bash
git clone <repository_url>
```

- Creates a copy of a remote repository on the local machine.

- Status

```bash
git status
```

- Shows the status of changes as untracked, modified, or staged.

- Add (Staging)

```bash
git add <file_name>
```

- Adds a file to the staging area, preparing it for the next commit.

- Add All (Staging)

```bash
git add .
```

- Adds all changes in the working directory to the staging area.

- Commit

```bash
git commit -m "Commit message"
```

- Records the changes in the staging area to the local repository.

- Commit All

```bash
git commit -am "Commit message"
```

- Adds and commits all changes in one command.

- Commit History

```bash
git log
```

- Displays the commit history, showing commit messages, authors, and timestamps.

- Diff

```bash
git diff
```

- Shows the differences between the working directory and the last commit.

- Reset (Unstage)

```bash
git reset <file_name>
```

- Unstages a file, removing it from the staging area.

- Reset (Commit)

```bash
git reset --soft HEAD^
```

- Resets the last commit, keeping changes in the staging area.

- Reset (Hard)

```bash
git reset --hard HEAD^
```

- Resets the last commit and discards changes in the working directory.

These basic Git commands are fundamental for managing changes, staging files, and making commits in a Git repository.

Git Log & Git Checkout

- Inspect Changes:

View Changes in Working Directory

```bash
git diff
```

- Shows the differences between the working directory and the last commit.

View Changes in Staging Area

```bash
git diff --staged
```

- Displays the differences between the staging area and the last commit.

View Commit Changes

```bash
git show <commit_hash>
```

- Inspects the changes introduced in a specific commit.

- Undo Changes:

Discard Changes in Working Directory

```bash
git checkout -- <file_name.java>
```

- Discards changes in the working directory for a specific Java file.

Unstage Changes

```bash
git reset <file_name.java>
```

- Unstages changes for a specific Java file, keeping modifications in the working directory.

Amend the Last Commit

```bash
git commit --amend -m "New commit message"
```

- Modifies the last commit, allowing changes to the commit message or added files.

Revert to a Specific Commit

```bash
git revert <commit_hash>
```

- Creates a new commit that undoes the changes made in a specific commit.

- Collaborating:

Fetch Changes from Remote Repository

```bash
git fetch
```

- Retrieves changes from the remote repository without merging.

Pull Changes from Remote Repository

```bash
git pull origin <branch_name>
```

- Fetches changes and merges them into the local branch.

Push Changes to Remote Repository

```bash
git push origin <branch_name>
```

- Sends committed changes to the remote repository.

Create a New Branch

```bash
git branch <new_branch_name>
```

- Creates a new branch for developing a new feature or fixing a bug.

Switch to a Branch

```bash
git checkout <branch_name>
```

- Switches to an existing branch.

Merge Branches

```bash
git merge <branch_name>
```

- Combines changes from one branch into the current branch.

Resolve Merge Conflicts

- Conflict resolution occurs if changes in different branches overlap. Use:

  ```bash
  git mergetool
  ```

  or manually edit conflicted files.