Git Training for Beginner Java Developers

Session 1: Introduction to Git and Version Control

Introduction to Version Control: Version control is a system that tracks and manages changes to files and code over time. It provides a history of changes, enables collaboration among developers, and facilitates the management of different versions of your project.

Benefits of Version Control:

- History Tracking: Version control systems maintain a complete history of changes made to files. This allows you to see who made what changes and when.
- Collaboration: Multiple developers can work on the same project simultaneously without overwriting each other's work. Version control ensures seamless integration of changes.
- Branching and Merging: Version control enables you to create branches, which are independent lines of development. This makes it easy to work on new features or bug fixes without affecting the main codebase.
- Revert to Previous Versions: If a mistake is made or a bug is introduced, version control allows you to roll back to a previous, working version of your code.
- Code Reviews: Version control systems help in conducting code reviews by allowing developers to share changes and provide feedback before integrating them into the main codebase.

Session 2: Basic Git Concepts and Initialization

Repositories, Commits, and Branches:

- A repository (or repo) is a directory that contains your project's files and their complete history.
- A commit represents a snapshot of your project at a specific point in time. It captures changes you've made to files.
- Branches are separate lines of development that allow you to work on features or fixes without affecting the main codebase.

Working Directory, Staging Area, and Git Repository:

- The working directory is where you modify your files.
- The staging area is where you prepare files for commit.
- The Git repository stores the complete history of commits.

Exercise: Participants set up Git on their machines and configure user information using the following commands:
Example:
git config --global user.name "Your Name" git config --global user.email "your@email.com"

Session 3: Basic Workflow and Status Checking
Basic Git Workflow:

1. Modify Files: Make changes to your project's files.
2. Stage Changes: Add the modified files to the staging area using git add.
3. Commit Changes: Create a commit containing the staged changes using git commit.

Checking Repository Status: The git status command shows you which files have been modified, which files are in the staging area, and which branch you're on.

Exercise: Participants practice the basic workflow on the Java project, creating a new class and committing it using the following commands:
Example:
touch MyFeature.java git status git add MyFeature.java git commit -m "Add new feature"
Real-World Scenario: Describe a scenario where participants need to track changes for a new feature.

Session 4: Branching and Merging
Branching and Creating a Branch:

- A branch is a parallel line of development. Use git branch to see existing branches and git branch <branch-name> to create a new branch.
- Switch between branches using git checkout <branch-name>.

Merging Branches:

- To integrate changes from one branch into another, use git merge. This combines the changes and creates a new commit.

Exercise: Participants create feature branches, make changes, and merge them back using the following commands:
Example:
git branch feature-branch git checkout feature-branch # Make changes and commit git checkout main git merge feature-branch
Real-World Scenario: Describe a collaborative project where different developers work on separate features.

Session 5: Collaborative Development with Remote Repositories

Introduction to Remote Repositories: A remote repository is a repository hosted on a server. It allows developers to collaborate and share code changes.
Working with Remote Repositories:

- git remote add <name> <repository-url>: Connect your local repository to a remote repository.
- git push <remote-name> <branch-name>: Push local commits to the remote repository.
- git pull <remote-name> <branch-name>: Pull changes from a remote repository.

Exercise: Participants create a remote repository on GitHub, push their local repository, and pull changes from the remote using the following commands:
Example:
# On GitHub: Create a new repository git remote add origin <repository-url> git push -u origin main # Make changes locally and push them git pull origin main
Real-World Scenario: Discuss how to fork and contribute to open-source Java projects.

Session 6: Resolving Merge Conflicts
Common Merge Conflicts: Merge conflicts occur when Git can't automatically combine changes from different branches. This often happens when two branches modify the same part of a file.
Resolving Merge Conflicts:

- Use git status to identify conflicted files.
- Manually edit the conflicted file to resolve conflicts.
- Use git add to mark resolved files as resolved.
- Commit the changes to complete the merge.

Exercise: Participants deliberately create merge conflicts, resolve them, and complete the merge using the following commands:
Example:
```
# In branch1: Modify a file git add modified_file git commit -m
"Modify file in branch1" git checkout main # In main: Modify the
same file git merge branch1 # Resolve conflicts, then commit
```
Interactive Element: Group discussion on participants' experiences with merge conflicts.

Session 7: Git Ignore and Best Practices
Introduction to .gitignore Files: A .gitignore file specifies files and directories that Git should ignore, such as build artifacts or temporary files.
Best Practices for Commit Messages:

- Write clear and concise commit messages.
- Start the message with a capitalized verb (e.g., "Add", "Fix", "Update").
- Keep the message under 72 characters and use the body for more details.

Exercise: Participants create a .gitignore file for their Java project and make commits adhering to best practices using the following commands:
Example:
```
touch .gitignore # Add files/directories to ignore in .gitignore git
add .gitignore git commit -m "Add .gitignore"
```
Real-World Scenario: Discuss how to manage sensitive information like API keys in a Git repository.

Session 8: Branching Strategies and Code Reviews

Different Branching Strategies:

- Feature Branching: Each new feature is developed in its own branch.
- Release Branching: Preparing a branch for release while development continues on other branches.
- Gitflow Workflow: A model with separate branches for features, releases, and hotfixes.

Using Git in Code Review Workflows:

- Developers create feature branches and submit pull requests for review.
- Reviewers provide feedback and discuss changes on the pull request.
- Changes are integrated into the main branch after approval.

Exercise: Participants follow a feature branching strategy for their Java project, make changes, and initiate a simulated code review using the following commands:

Example:

git checkout -b feature/add-login # Make changes and commit git push origin feature/add-login # Initiate a code review and discuss changes

Real-World Scenario: Present a scenario where a new feature needs to be developed and reviewed.

<u>Hands'on</u>

git add .	Adds all the changes to git staging (Pre commit)

git add F3.java	Adds F3.java file to the git staging (Pre commit)
git add SampleFile.java	adds SampleFile.java to the git staging
git add src/main/java/FirstController.java	
git branch	lists down all the branches in the current git repository
git branch feature-branch-1	creates a branch named feature-branch-1
git branch feature-branch-2	
git checkout feature-branch-1	switches to the branch named feature-branch-1
git checkout feature-branch-2	switches to the branch named feature-branch-2
git checkout master	switches to the master branch. Post this, working branch would be master
git clone https://user@bitbucket.org/project/training.git	clones the remote repository into user's current working directory
git commit -m"feat : initial commits"	Git commit by providing a commit message
git commit -m"fix: hotfix on release-1"	
git init	makes the current directory a git repository (LOCAL)

git log	shows the commit history of user's working branch
git merge feature-branch-1	merges the feature-branch-1 branch with user's working branch (ex :master)
git merge feature-branch-2	merges the feature-branch-2 branch with user's working branch (ex :master)
git pull	Pulls the changes from remote repository(origin) to local
git push	Pushes the changes (commits made locally) into remote repository
git restore SampleFile.java	the changes made locally can be restored to the previously commit state
git rm --cached 1.bak	removes the file 1.bak from git staging area (Pre commit)
git rm --cached SampleFile.java	removes the file SampleFile.java from git staging area (Pre commit)
git status	It displays the user's local repository status