Student Handout: Git Integration with Jenkins

# 1. Role of Git in Jenkins Pipelines

Git provides the foundation for modern CI/CD pipelines by:

- Version Control: Tracks changes to application source code.

- Collaboration: Enables teams to work together efficiently.

- Triggers Builds: Automatically starts Jenkins builds on code changes.

- History: Maintains a complete record of commits.

- Branching Strategies: Supports feature, test, and production workflows.

Why it matters: Git ensures Jenkins always builds from the latest committed code.

# 2. Installing Git on Jenkins Server

Verify Java:

- Jenkins requires Java (11 or 17 recommended).

Install Git:

- Linux:

sudo apt-get install git

# or

sudo yum install git

- Windows: Download from git-scm.com

Verify Installation:

git --version

Ensure jenkins user can run Git. On Windows, confirm Git is in the system PATH.

# 3. Configuring Git in Jenkins

1. Open Jenkins Dashboard.

2. Go to Manage Jenkins → Global Tool Configuration.

3. Scroll to Git section → click Add Git.

4. Provide:

- Name: Default Git

- Path to Git executable

5. Save.

Jenkins can now use Git for all jobs.

# 4. Jenkins + Git Workflow

1. Clone → Jenkins clones the repo.

2. Build → Code is compiled/packaged.

3. Test → Automated tests run.

4. Deploy → App is deployed to target environment.

This ensures CI/CD from commit → deployment with minimal manual work.

# 5. Creating a Freestyle Job with Git

1. Dashboard → New Item → Freestyle project.

2. In Source Code Management, select Git.

3. Enter repository URL (HTTPS or SSH).

4. Choose branch to build (e.g., \*/main).

5. Add build triggers (e.g., poll SCM).

6. Add build steps (e.g., shell commands).

7. Save & run job.

Good for simple integrations before moving to Pipelines.

# 6. Advanced Git Configuration

- Credentials Management → Use Jenkins Credentials Plugin for private repos.

- Branch Specifiers → Build specific branches (e.g., \*/feature/\*).

- Submodules → Enable recursive clone for nested repos.

- Shallow Clone → Limit history for faster builds.

# 7. Webhook Integration with Git Providers

Webhooks = automatic job triggers when code is pushed.

Supported Providers:

- GitHub

- GitLab

- Bitbucket

GitHub Webhook Setup:

1. In Jenkins:

- Install GitHub Integration plugin.

- Configure job with GitHub hook trigger for GITScm polling.

2. In GitHub:

- Repo → Settings → Webhooks → Add webhook.

- Payload URL: http://your-jenkins-url/github-webhook/

- Content type: application/json.

3. Test Connection → Verify Jenkins logs show webhook received.

# 8. Demo: Git Push to Jenkins Build

1. Developer pushes code to Git.

2. GitHub/GitLab sends webhook → Jenkins.

3. Jenkins auto-triggers job.

4. Job builds, tests, and packages code.

5. Status sent back to Git provider.

No manual job triggering needed → faster feedback & automation.

# 9. Key Takeaways

- Seamless Integration: Git + Jenkins = automated CI/CD.

- Simple Setup: Minimal config, big productivity gains.

- Webhooks = Automation: Push → build → test → deploy.

# 10. Next Steps

- Store a Jenkinsfile in your repo → Pipeline as Code.

- Add branching strategies (feature, release, hotfix).

- Explore advanced plugins (Multibranch Pipeline, GitHub Branch Source).