Jenkins Pipeline: Step-by-Step Detailed Explanation

1. Introduction to Jenkins Pipeline

Jenkins Pipeline is a powerful automation tool that allows you to define your entire CI/CD process as code using Groovy.

It replaces the manual configuration of Freestyle projects, making automation reproducible, version-controlled, and

error-free. Pipelines provide better workflow management and support complex multi-stage processes.

2. Pipeline Structure

A Pipeline has three main components:

- pipeline: the root block.
- agent: defines where the code will run (e.g., any node).
- stages: contains multiple stages like Build, Test, Deploy.
- stage: represents one phase of execution.
- steps: define actual commands to execute.

Example:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'echo Building...'
            }
        }
    }
}
```

3. Creating a Basic Pipeline

Steps:

1. Go to Jenkins -> New Item -> Select "Pipeline"
2. Write your pipeline script under the Pipeline Script section
3. Use 'sh' for Linux commands or 'bat' for Windows commands
4. Save and Build

Example:

```
pipeline {
    agent any
    stages {
        stage('Git Clone') {
            steps {
                sh 'git clone https://github.com/DevSecOpsG/helloworldsimplejavaprogram'
            }
        }
        stage('Build') {
            steps {
                sh 'javac helloworld.java'
            }
        }
        stage('Run') {
            steps {
                sh 'java Simple'
            }
        }
    }
}
```

4. Pipeline Syntax Generator

The Syntax Generator helps generate correct code snippets for Jenkins pipeline steps.

Steps:

- Go to your Jenkins job -> Pipeline Syntax -> Choose step type (e.g., git, sh)
- Fill parameters and click "Generate Pipeline Script"
- Copy the generated code into your Jenkinsfile

Example:

```
checkout([$class: 'GitSCM', branches: [[name: '*/main']],
  userRemoteConfigs: [[url: 'https://github.com/DevSecOpsG/helloworldsimplejavaprogram']]
])
```

5. Git Integration

Public repositories do not need credentials. Private ones require Jenkins credentials setup.

Public Repo Example:

```
git branch: 'main', url: 'https://github.com/DevSecOpsG/helloworldsimplejavaprogram'
```

Private Repo Example:

```
git credentialsId: 'my-cred', branch: 'main', url: 'https://github.com/private/repo.git'
```

6. Setting Up CI Trigger

Triggers automate builds. You can poll SCM or use Webhooks.

Example:

```
pipeline {
    triggers {
        pollSCM('* * * * *') // every minute
    }
}
```

7. Parameterized Builds

Allow user input for dynamic environments (Dev, QA, Prod).

Example:

```
pipeline {
    parameters {
        choice(name: 'ENV', choices: ['DEV', 'QA', 'PROD'], description: 'Choose environment')
    }
    stages {
        stage('Deploy') {
            steps {
                sh "echo Deploying to ${params.ENV}"
            }
        }
    }
}
```

8. Post-Build Actions

Post block defines what happens after build success/failure.

Example:

```
pipeline {
    post {
        success {
            echo 'Build completed successfully!'
            build job: 'DeployJob'
        }
        failure {
            echo 'Build failed!'
        }
    }
}
```

9. Windows vs Linux Differences

| Feature | Linux | Windows |

|---|---|---|

| Command | sh | bat |

| Paths | /home/ubuntu | C:\Users |

| Package Manager | apt-get | choco |

| Env Setup | Auto | Manual PATH |

Example:

sh 'javac HelloWorld.java' // Linux

bat 'javac HelloWorld.java' // Windows

Homework:

Convert your previous Freestyle project into a Pipeline project with the following stages:

1. Git Clone
2. Build
3. Run

This builds understanding of Groovy scripting, automation, and CI/CD principles.