Devops[Development and Operations]: CIAEC59

Unit-2

Unit II

- Build Tools: Git- Git Basics, Repository, Branching / Merging, Git Workflow. Maven, - Maven, POM (Project Object Model, Gradle -Gradle, Groovy / Kotlin DSL, Build Lifecycle, Dependency Management, Build Automation.

Build Tools-Git

Git is an open-source distributed version control system that helps teams track and manage code changes, collaborate seamlessly, and work on projects of any size. It keeps a history of every change, allowing you to revisit or restore previous versions, and makes it easy to fix mistakes without losing progress.

RAMAIAH
Institute Of Technology

Build Tools-Git

DEVOTION TO ENLIGHTENMENT

Core Concepts of Git

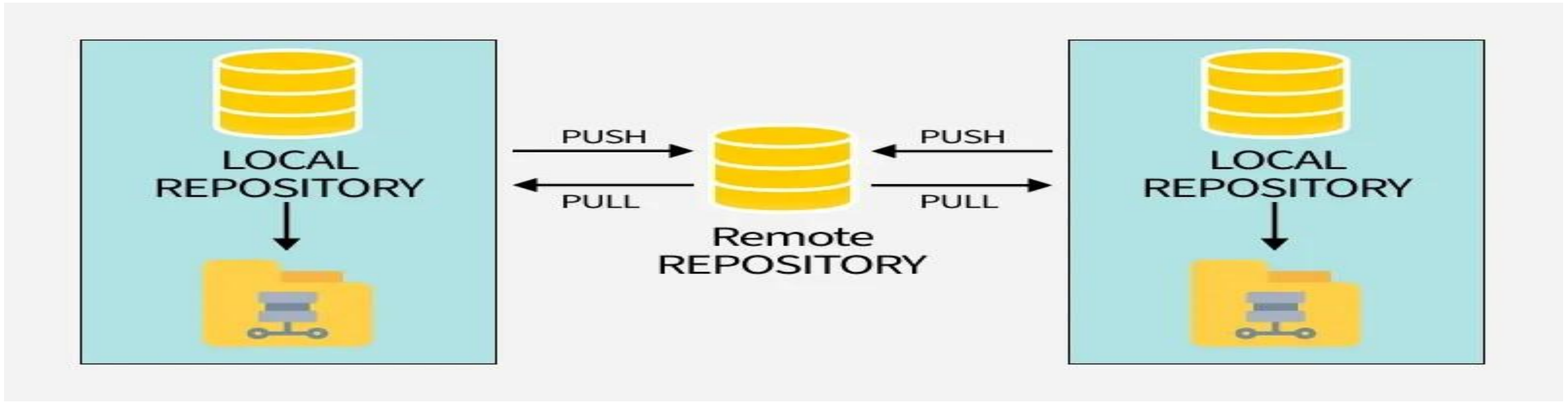
Before using Git, it is important to understand some of its core concepts. These concepts will help you get started and make it easier to work with Git in real-world scenarios.

1. Repositories

A repository (or repo) is a storage space where your project files and their history are kept. There are two types of repositories in Git:

Local Repository: A copy of the project on your local machine.

Remote Repository: A version of the project hosted on a server, often on platforms like GitHub, GitLab, or Bitbucket.



Build Tools-Git

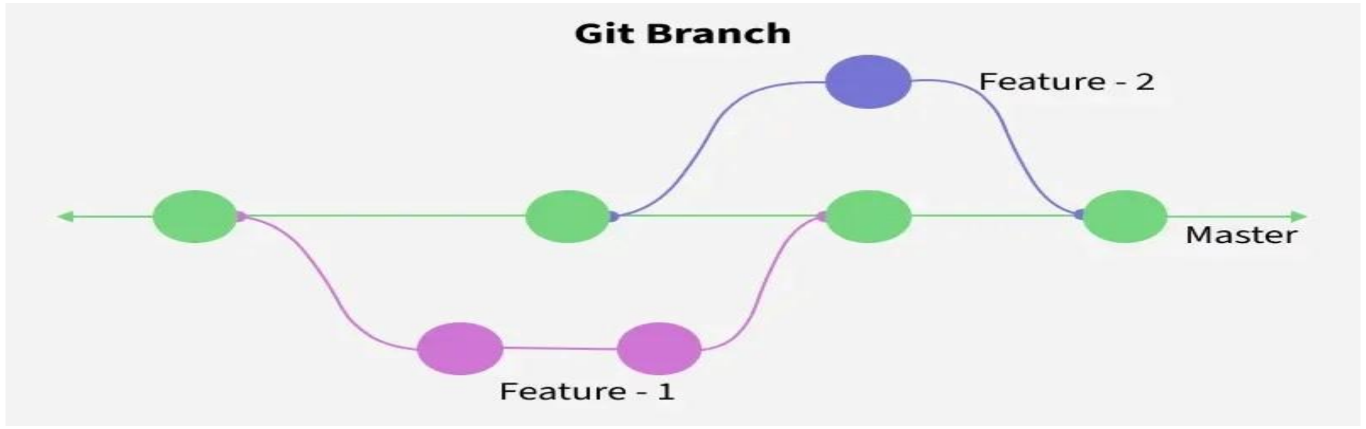
A commit is a snapshot of your project at a specific point in time. Each commit has a unique identifier (hash) and includes a message describing the changes made. Commits allow you to track and review the history of your project.

Branches

Branches allow developers to work on separate tasks without affecting the main codebase. Common branch types include:

Main (or Master) Branch: The stable version of the project, usually production-ready.

Feature Branch: Used for developing new features or bug fixes.



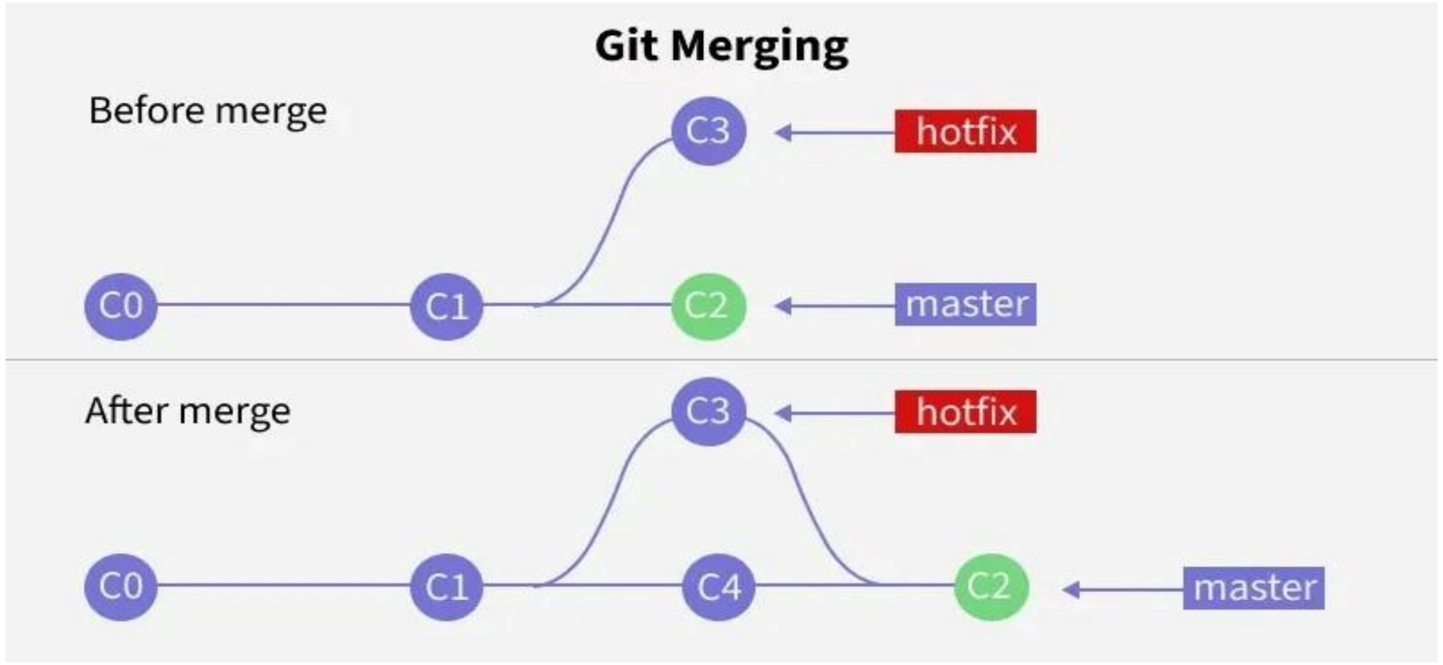
RAMAIAH Institute Of Technology

Build Tools-Git

DEVOTION TO ENLIGHTENMENT

Merging

Merging is the process of integrating changes from one branch into another. It allows you to combine the work done in different branches and resolve any conflicts that arise.



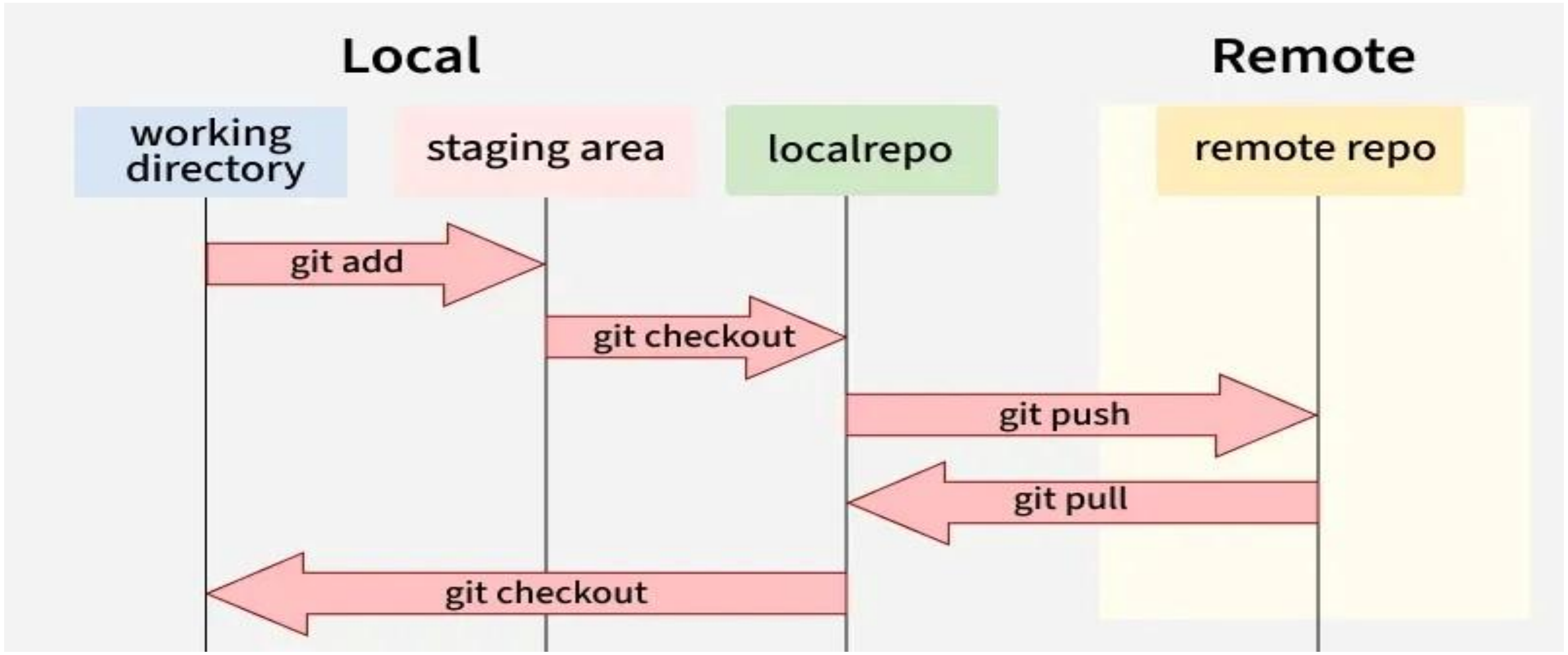
Build Tools-Git

Basic Git Commands

Command	Description
git status	Shows the current status of the repository — staged, unstaged, and untracked files.
git add <file-name>	Stages a specific file for commit. Use `git add .` to stage all changes.
git commit -m "message"	Commits the staged changes with a descriptive commit message.
git branch <branch-name>	Creates a new branch with the given name.
git checkout <branch-name>	Switches to the specified branch.
git merge <branch-name>	Merges changes from the given branch into the current branch.
git push origin <branch-name>	Pushes the local branch changes to the remote repository.
git pull origin <branch-name>	Fetches and merges changes from the remote repository into the local branch.
git log	Displays the commit history for the current branch.

Build Tools-Git

- **Git Workflow**



Build Tools-Git

Git Workflow: Git workflows define how developers should use Git in a structured and efficient manner.

1. Clone the Repository

```
git clone git@github.com:username/repository.git
```

2. Create and Switch to a New Branch

```
git checkout -b feature-branch
```

3. Make Changes and Stage Them

```
git add <file-name>
```

4. Commit the Changes

```
git commit -m "Add new feature"
```

5. Push the Changes

```
git push origin feature-branch
```

6. Create a Pull Request: After pushing your changes, create a pull request on GitHub to merge the feature branch into the main branch.

7. Update your Local Repository

```
git checkout main
git pull origin main
```

8. Delete Feature Branch:

```
git branch -d feature-branch
git push origin --delete feature-branch
```

Build Tools-Maven

Maven is a build automation tool developed using the Java programming language. It is primarily used for Java-based projects to manage the build process, including source code compilation, testing, packaging, and more. Maven utilizes the Project Object Model (POM), where the pom.xml file describes the project's configuration and dependency management.

Features of Maven:

The Maven Build Automation tool provides a lot of features to make the development easy. Below we listed them

Dependency Management: Automatically downloads and manages external libraries.

Standard Project Structure: Follows a fixed folder layout for source, test, and other files.

Build Lifecycle: Defines standard build phases like compile, test, and deploy.

Plugins: Supports plugins for compiling, testing, packaging, and more.

POM File: Uses pom.xml to manage configuration and dependencies.

Central Repository: Fetches dependencies from a shared online repository.

Build Profiles: Supports different settings for dev, QA, and production.

Reporting: Can generate Javadoc, test reports, and project documentation.

IDE Support: Integrates with Eclipse, IntelliJ, NetBeans, etc.

Build Tools-Maven

Maven provides a standard Project folder structure you can observe this in the above image.

src/main/java: It contains the main Java source code.

src/main/resources: It contains non-Java resources used by the application.

src/main/webapp: It contains resources for web applications.

src/test/java: It contains test source code.

src/test/resources: It contains resources used for testing.

target: It contains compiled classes, packaged JARs/WARs, and other built artifacts.

pom.xml: The Project Object Model file that defines the project configuration, dependencies, and build settings.

Maven Build Life Cycle

In Maven Build life cycle there are different phases these phases are used for different purposes like test phase for complies tests, verify phases is used for tests and main source code and other.

Validate: This phase is responsible for validates if the project structure is correct or not.

Compile: It compiles the source code, converts the .java files to .class, and stores these classes in the target/classes folder.

Test: It runs unit tests for the maven project

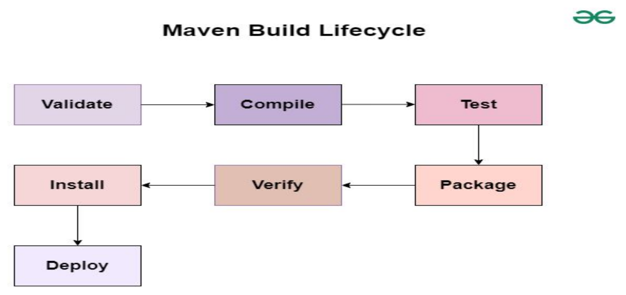
Package: This phase is responsible for distribute complied code in the format of WAR, JAR and others.

Integration test: It runs the integration tests for the maven project

Verify: verify that the project is valid and meets the quality standards.

Install: This phase is responsible for install packaged code on the system.

Deploy: copies the packaged code to the remote repository for deployment then other developers can easily access this one.



Build Tools-Maven

1. Git – Version Control Basics

Git is a distributed version control system used to track changes in source code during software development.

1.Git Basics

Repository (Repo): A repository is a storage location for your project code.

Local Repo: Stored on your machine.

Remote Repo: Stored on servers like GitHub, GitLab, or Bitbucket.

Cloning a repo: Copying a remote repository locally.

2.git clone <repository_url>

Check Status: Shows changes in the working directory.

3.git status

Add Changes: Stages changes for commit.

git add <file_name> # Or 'git add .' for all files

4.Commit Changes: Saves staged changes locally.

git commit -m "Commit message"

5.Push Changes: Uploads commits to the remote repository.

git push origin <branch_name>

6.Pull Changes: Fetches and merges updates from remote.

git pull

Branching & Merging

Build Tools

Branching & Merging

Branch: A separate line of development.

git branch <branch_name>

git checkout <branch_name> # Switch branch

Merge: Combine changes from one branch to another.

git checkout main

git merge <feature_branch>

Conflict Resolution: Occurs when changes clash. Must manually edit, then add & commit.

Git Workflow

1 Feature Branch Workflow:

Create a branch → Develop → Commit → Merge to main.

2. Fork & Pull Workflow:

Fork repo → Clone → Develop → Pull request to original repo.

3. Gitflow Workflow:

Main, develop, feature, release, and hotfix branches structured for larger projects.

Build Tools

2. Maven – Project Build & Dependency Management

Maven is a build automation tool primarily for Java projects.

Key Concepts

POM (Project Object Model): pom.xml defines the project's:

Dependencies

Plugins

Build configurations

Project info (groupId, artifactId, version)

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0.0</version>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>5.3.10</version>
    </dependency>
  </dependencies>
</project>
```

Build Tools

Maven Lifecycle

Clean: Deletes target directory.

mvn clean

Compile: Compiles the source code.

mvn compile

Test: Runs unit tests.

mvn test

Package: Packages code into JAR/WAR.

mvn package

Install: Installs package to local repository.

mvn install

Deploy: Uploads artifacts to remote repository.

Dependency Management

- Maven automatically downloads required libraries from central repositories.
- Dependencies are defined in pom.xml.

Build Tools-Gradle

Gradle is an open-source construction tool that is capable of controlling the development tasks with compilation and packaging, including testing, deployment and publishing.

It is an automation tool that is based on Apache Ant and Apache Maven.

This tool is capable of developing applications with industry standards and supports a variety of languages, including Groovy, C++, Java, Scala and C.

Working of Gradle

The Gradle project, when constructed it consists of one or more projects. These projects consist of tasks. Let us understand the basics of both terms.

1. Gradle Projects: The projects created by Gradle are a web application or a JAR file. These projects are a combination of one or more tasks. These projects are capable of being deployed on various development life cycles. A Gradle project can be described as building a wall with bricks N in number, which can be termed as tasks.

Build Tools-Gradle

2. Gradle Tasks: The tasks are the functions that are responsible for a specific role. These tasks are responsible for creating classes, Javadoc, or publishing archives into the repository, which makes up the whole development of the Gradle project. These tasks help Gradle decide what input is to be processed for a specific output. Again, tasks can be categorized in two different ways:

Default Task: These are the predefined tasks that are provided to users by Gradle. These are provided to users prior which executing when the users do not declare any task on their own. For example, init and wrap the default tasks provided to users into a Gradle project

Custom Task: Custom tasks are the tasks that are developed by the developer to perform a user-defined task. These are developed to run a specific role in a project. Let's take a look at how to develop a Custom Task below.

Example: Printing Welcome Gradle! with a task in Gradle.

```
build.gradle : task hello
{
   doLast
   {
      println 'Welcome to Gradle!'
   }
}
```

Output:

```
> gradle -q hello
Welcome to Gradle!
```

Build Tools-Gradle

Fetaures of Gradle:

IDE Support: Compatible with popular IDEs, making development seamless across environments.

Java Compatibility: Requires the JVM to run and supports Java APIs, making it familiar for Java developers.

Build System Integration: Supports features from Ant and Maven, including importing Ant projects and using Maven repositories.

Incremental Builds: Only recompiles code that changed since the last build, reducing build time.

Open Source: Free and open-source under the Apache License with strong community support.

Multi-Project Builds: Efficiently handles complex project structures with multiple sub-projects.

Dependency Management: Automatically resolves and downloads project dependencies.

Scripting with Groovy/Kotlin DSL: Offers flexible scripting for build configurations.

Plugins: Offers a wide range of plugins to support various languages and technologies like Java, Android, C++, etc.

Extensibility: Highly customizable through APIs and plugin development.

Build Caching: Speeds up builds by reusing outputs from previous builds.

Test Automation: Supports testing frameworks like JUnit, TestNG and Spock, with code coverage tools.

Continuous Integration: Easily integrates with CI tools like Jenkins and TeamCity.

Multi-Language Support: Supports Java, Groovy, Kotlin, Scala and more.

Build Tools-Gradle

Pros And Cons of Gradle:

Declarative and Scalable: Uses a clear DSL for configuration and scales well with project size.

Flexible Structure: Adapts to any project layout and supports custom plugins.

Deep API Access: Allows detailed control over build execution and behavior.

Improved Performance: Optimized for faster builds, even in large projects.

Strong Community: Offers rich documentation, tutorials and plugin resources.

Cons of Using Gradle

Learning Curve: Requires knowledge of Groovy/Java and an understanding of Gradle's architecture.

Complex Configuration: Setup and plugin integration can be tricky for beginners.

Debugging Difficulty: Troubleshooting can be hard in large builds with many dependencies.

Resource Intensive: Can consume significant system resources during builds.

Migration Challenges: Transitioning from other build tools may require significant effort and expertise.

Build Tools

2.Gradle-Init

```
rit@rit:~$ gradle init
openjdk version "11.0.28" 2025-07-15
OpenJDK Runtime Environment (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1, mixed mode, sharing)
Starting a Gradle Daemon (subsequent builds will be faster)

BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 executed
```

RAMAIAH Institute Of Technology

Build Tools

DEVOTION TO ENLIGHTENMENT

2.Gradle-Init

```
rit@rit:~$ gradle init
openjdk version "11.0.28" 2025-07-15
OpenJDK Runtime Environment (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1)
OpenJDK 64-Bit Server VM (build 11.0.28+6-post-Ubuntu-1ubuntu124.04.1, mixed mode, sharing)
Starting a Gradle Daemon (subsequent builds will be faster)
 BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 executed
```

Groovy DSL (Domain Specific Language)

Overview

Gradle was originally designed using the **Groovy language**, a dynamic JVM-based scripting language.
The **Groovy DSL (Domain-Specific Language)** allows developers to define build configurations in a declarative and flexible manner using Groovy syntax inside a file named build.gradle.

It serves as the **core language** for describing how a project should be built, tested, and packaged.

Key Features

Dynamic Typing: No need to declare data types explicitly.

Scripting Flexibility: Combines declarative configuration with imperative logic.

Concise Syntax: Less verbose compared to XML-based tools like Maven.

Easy Integration: Works seamlessly with Java and other JVM-based tools.

RAMAIAH Institute Of Technology

Build Tools

DEVOTION TO ENLIGHTENMENT

Example: build.gradle

```
plugins {
    id 'java'
}
group = 'com.example'
version = '1.0.0'
 repositories {
    mavenCentral()
}
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter'
    testImplementation 'junit:junit:4.13.2'
}
 task hello {
    doLast {
        println 'Hello from Groovy DSL!'
    }
}
```

Explanation

The plugins block defines applied plugins.

The repositories block specifies where dependencies are fetched from.

The dependencies block lists libraries required for build and runtime.

The task block defines custom build tasks.

Build Tools

Advantages

- High flexibility and easy customization.
- Ideal for users familiar with Groovy syntax.
- Widely supported and compatible with all Gradle features.

2. Kotlin DSL

Overview

The Kotlin DSL (build.gradle.kts) is a modern, type-safe, and statically typed alternative to the Groovy-based build scripts.

Introduced by Gradle in recent versions, Kotlin DSL provides improved code completion, error highlighting, and compile-time validation in IDEs such as IntelliJ IDEA or Android Studio.

Key Features

- Static Typing: Compile-time validation prevents runtime script errors.
- IDE Support: Excellent auto-completion and refactoring capabilities.
- Safer Syntax: Reduces syntax ambiguity common in Groovy DSL.
- Interoperable: Fully compatible with existing Groovy DSL configurations.

Build Tools

Example: build.gradle.kts

```
plugins {
   id("java")
}
group = "com.example"
version = "1.0.0"

repositories {
   mavenCentral()
}
dependencies {
   implementation("org.springframework.boot:spring-boot-starter")
   testImplementation("junit:junit:4.13.2")
}
tasks.register("hello") {
   doLast {
      println("Hello from Kotlin DSL!")
   }
}
```

Advantages

- Strong type-checking and IDE validation.
- Better maintainability and readability for large projects.
- More consistent and predictable than Groovy scripts.

Build Tools

Gradle Build Lifecycle

Gradle executes every build in three sequential phases. Understanding this lifecycle helps developers structure their build scripts effectively.

Phases

1. Initialization Phase

- Determines which projects are involved in the build.
- Creates Project objects for each module in multi-project builds.
- Executes settings.gradle or settings.gradle.kts to configure included projects.

2. Configuration Phase

- Reads and configures all tasks in every project.
- Executes the code inside the build scripts.
- Builds the task dependency graph (but doesn’t execute tasks yet).

3. Execution Phase

- Executes only the tasks requested on the command line (e.g., gradle build).
- Executes dependencies automatically in the correct order.

Example Build Flow

gradle clean build

Execution order:

1. clean – Deletes previous build directories.
2. compileJava – Compiles source code.
3. processResources – Copies resources.
4. test – Runs unit tests.
5. jar – Packages compiled classes.
6. build – Assembles the final artifact.

RAMAIAH Institute Of Technology

Build Tools

DEVOTION TO ENLIGHTENMENT

Lifecycle Diagram

Initialization → Configuration → Execution

(settings.gradle)   (build.gradle)   (task execution)

Benefits

- Modular and flexible.
- Clear control over task dependencies.
- Efficient — executes only necessary tasks.

📦 4. Dependency Management

Overview

Gradle's dependency management system handles external libraries and modules used in the project.

Dependencies are resolved from repositories like Maven Central, Google, or local repositories.

Components of Dependency Management

1 **Repositories**

o Specify the source location for dependencies.

o Common examples:

o repositories {

o   mavenCentral()

o   google()

o }

2. **Dependency Configurations**

o Define the scope and purpose of dependencies:

- implementation: Used for main source code.
- api: Exported dependencies visible to consumers.
- compileOnly: Needed only during compilation.
- runtimeOnly: Used at runtime.
- testImplementation: For test classes.

Build Tools

3. Dependency Notation

- Syntax: group:artifact:version
- Example:

```
dependencies {
    implementation 'org.apache.commons:commons-lang3:3.12.0'
    testImplementation 'junit:junit:4.13.2'
}
```

4. Transitive Dependencies

- Gradle automatically includes dependencies of dependencies.
- Example: If Library A depends on Library B, both are included.

5. Dependency Resolution

- Gradle checks version conflicts and selects compatible versions.
- Allows dependency locking for reproducible builds.

Advantages

- Simplifies library management.
- Supports caching and offline builds.
- Resolves version conflicts automatically.
- Integrates seamlessly with Maven/Ivy repositories.

RAMAIAH Institute Of Technology

Build Tools

DEVOTION TO ENLIGHTENMENT

5. Build Automation

Overview

Build automation refers to the process of automatically compiling, testing, packaging, and deploying code using build tools like Gradle.

It eliminates manual intervention and ensures consistency across environments.

Common Automated Tasks

Task	**Description**
gradle clean	Deletes old build artifacts.
gradle compileJava	Compiles source files.
gradle test	Runs all test cases.
gradle jar	Packages compiled code into JAR files.
Gradle build	Performs full build including compilation and testing.

Continuous Integration (CI)

Gradle integrates with CI/CD tools such as:

- Jenkins
- GitHub Actions
- GitLab CI/CD
- CircleCI

Build Tools

DEVOTION TO ENLIGHTENMENT

Example CI command:

./ **gradlew clean test build**

Benefits of Build Automation

- Reduces human errors.
- Ensures reproducible builds.
- Increases productivity and consistency.
- Enables continuous delivery pipelines.

Example Jenkinsfile Snippet

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh './gradlew clean build'
      }
    }
    stage('Test') {
      steps {
        sh './gradlew test'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying application...'
      }
    }  }
}
```

Thank
You