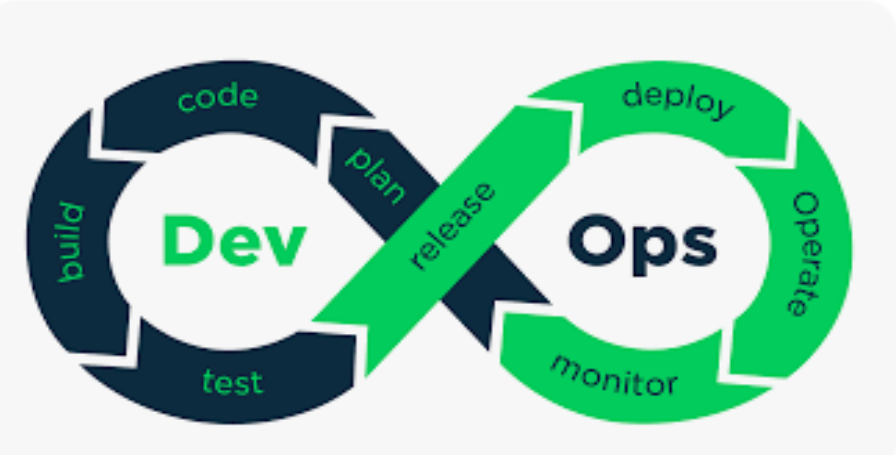

Experiment – 1: Working with Maven: Creating a Maven Project, Understanding the POM File

Experiment – 2: Installing and Setting Up Gradle in IntelliJ IDEA

Experiment – 3: Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts

Experiment – 4: Gradle Kotlin DSL: Setting Up & Building a Kotlin Project in IntelliJ IDEA

Experiment – 5: Build and Run a Java Application with Maven, Migrate the Same Application to Gradle

Experiment – 6: Understanding and Working with Jenkins

Experiment - 1
Working with Maven: Creating a Maven Project, Understanding the POM File

Steps to Create a Maven Project in IntelliJ IDEA

1. **Install Maven (if not already installed):**
   - Download Maven from the official website.
   - Set the `MAVEN_HOME` environment variable and update the system `PATH`.
2. **Create a New Maven Project:**
   - Open IntelliJ IDEA.
   - Go to `File` > `New` > `Project`.
   - Select `Maven` from the project types.
   - Choose `Create from Archetype` (optional) or proceed without.
   - Set the project name and location, then click `Finish`.
3. **Set Up the `pom.xml` File:**
   - The `pom.xml` file is where you define dependencies, plugins, and other configurations for your Maven project.
   - Example of a basic `pom.xml`:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>simple-project</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <!-- Add your dependencies here -->
    </dependencies>

</project>

```

4. **Add Dependencies for Selenium and TestNG:**
   - In the `pom.xml`, add Selenium and TestNG dependencies under the `<dependencies>` section.

```
<dependencies>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.141.59</version>
    </dependency>
    <dependency>
        <groupId>org.testng</groupId>
        <artifactId>testng</artifactId>
        <version>7.4.0</version>
        <scope>test</scope>
    </dependency>
</dependencies>

```

5. **Create a Simple Website (HTML, CSS, and Logo):**
   - In the `src/main/resources` folder, create an `index.html` file, a `style.css` file, and place the `logo.png` image.

Example of a simple `index.html`:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My Simple Website</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <img src="logo.png" alt="Logo">
    </header>
    <h1>Welcome to My Simple Website</h1>
</body>
</html>

```

6. **Upload the Website to GitHub:**
   - Initialize a Git repository in your project folder:

```
git init

```

   - Add your files and commit them:

```
git add .
git commit -m "Initial commit"

```

   - Create a GitHub repository and push the local project to GitHub:

```
git remote add origin <your-repository-url>
git push -u origin master

```

Experiment - 2

Installing and Setting Up Gradle in IntelliJ IDEA

- **Creating a Gradle Project in IntelliJ IDEA**

Step 1: Open IntelliJ IDEA and Create a New Project

1. Click on **"New Project"**.
2. Select **"Gradle"** (under Java/Kotlin).
3. Choose **Groovy or Kotlin DSL (Domain Specific Language)** for the build script.
4. Set the **Group ID** (e.g., `com.example`).
5. Click **Finish**.

Step 2: Understanding Project Structure

```
my-gradle-project
|— build.gradle  (Groovy Build Script)
|— settings.gradle
|— src
|   ├— main
|   |   ├— java
|   |   └— resources
|   ├— test
|   |   ├— java
|   |   └— resources
```


- **Build and Run a Simple Java Application**

Step 1: Modify `build.gradle` (Groovy DSL)

```
plugins {
    id 'application'
}

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.8.1'
}

application {
    mainClass = 'com.example.Main'
}

```

Step 2: Create `Main.java` in `src/main/java/com/example`

```
package com.example;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello from Gradle!");
    }
}

```

Step 3: Build and Run the Project

- **In IntelliJ IDEA**, open the **Gradle tool window** (View → Tool Windows → Gradle).
- Click `Tasks > application > run`.
- Or run from terminal:

```
gradle run
```

Hosting a Static Website on GitHub Pages

Step 1: Create a `/docs` Directory

- Create `docs` inside the **root folder** (not in `src`).
- Add your **HTML, CSS, and images** inside `/docs`.

Step 2: Modify `build.gradle` to Copy Website Files (This is optional)


```
task copyWebsite(type: Copy) {
    from 'src/main/resources/website'
    into 'docs'
}

```


Step 3: Commit and Push to GitHub

```
git add .
git commit -m "Deploy website using Gradle"
git push origin main

```

Step 4: Enable GitHub Pages

- Go to **GitHub Repo → Settings → Pages**.
- Select the `/docs` **folder** as the source.

Your website will be hosted at:

```
https://yourusername.github.io/repository-name/
```

Experiment - 3

Working with Gradle: Setting Up a Gradle Project, Understanding Build Scripts

Setting Up the Gradle Project

1. Click on **"New Project"**.
2. Select **"Gradle"** (under Java/Kotlin).
3. Choose **Groovy or Kotlin DSL (Domain Specific Language)** for the build script.
4. Set the **Group ID** (e.g., `com.example`).
5. Click **Finish**.

- **Understanding Project Structure**

```
my-gradle-project
├── build.gradle  (Groovy Build Script)
├── settings.gradle
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   ├── test
│   │   ├── java
│   │   └── resources
```

- **Build and Run a Simple Java Application**

Step 1: Modify `build.gradle` (Groovy DSL)

```
plugins {
    id 'application'
}

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter:5.8.1'
}

application {
    mainClass = 'com.example.Main'
}

```

Step 2: Create `Main.java` in `src/main/java/com/example`

```
package com.example;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello from Gradle!");
    }
}

```

Step 3: Build and Run the Project

- **In IntelliJ IDEA**, open the **Gradle tool window** (View → Tool Windows → Gradle).
- Click `Tasks > application > run`.
- Or run from terminal:

```
gradle run
```

- **Hosting a Static Website on GitHub Pages**

Step 1: Create a `/docs` Directory

- Create `docs` inside the **root folder** (not in `src`).
- Add your **HTML, CSS, and images** inside `/docs`.

Step 2: Modify `build.gradle` to Copy Website Files (This is optional)


```
task copyWebsite(type: Copy) {
    from 'src/main/resources/website'
    into 'docs'
}

```


Step 3: Commit and Push to GitHub

```
git add .
git commit -m "Deploy website using Gradle"
git push origin main
```

Step 4: Enable GitHub Pages

- Go to **GitHub Repo → Settings → Pages**.
- Select the `/docs` **folder** as the source.

Your website will be hosted at:

```
https://yourusername.github.io/repository-name/

```

Experiment - 4

Gradle Kotlin DSL: Setting Up & Building a Kotlin Project in IntelliJ IDEA

1️⃣ Setting Up the Gradle Project

Step 1: Create a New Project

1. Open **IntelliJ IDEA**.
2. Click on **File > New > Project**.
3. Select **Gradle** as the build system.
4. Choose **Kotlin** as the language.
5. Select **Gradle Kotlin DSL** (it will generate `build.gradle.kts`).
6. Name your project (e.g., `MVNGRDKOTLINDEMO`).
7. Set the **JDK** (use **JDK 17.0.4**, since that's your version).
8. Click **Finish**.

2️⃣ Understanding `build.gradle.kts`

After creating the project, the default `build.gradle.kts` file looks like this:

```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile

plugins {
    kotlin("jvm") version "1.8.10"  // Use latest stable Kotlin version
    application
}

group = "org.example"
version = "1.0-SNAPSHOT"

```

```
repositories {
    mavenCentral()
}

dependencies {
    implementation(kotlin("stdlib"))  // Kotlin Standard Library
    testImplementation("org.junit.jupiter:junit-jupiter-api:5.8.2")
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:5.8.2")
}

tasks.test {
    useJUnitPlatform()
}

tasks.withType<KotlinCompile> {
    kotlinOptions.jvmTarget = "17"  // Match with your JDK version
}

application {
    mainClass.set("MainKt")  // Update this if using a package
}
```


3 Creating the Main Kotlin File

Now, create your `Main.kt` file inside `src/main/kotlin/`.

If you're using a package (e.g., `org.example`), it should look like:


```
package org.example

fun main() {
    println("Hello, Gradle with Kotlin DSL!")
}

```


If you're **not** using a package, remove the `package` line and ensure `mainClass.set("MainKt")` in `build.gradle.kts`.

4 Building and Running the Project

Build the Project

```
./gradlew build

```

Run the Project

```
./gradlew run

```

---

5 Packaging as a JAR

To run the project without IntelliJ, we need a **JAR file**.

Step 1: Create a Fat (Uber) JAR

Modify `build.gradle.kts`:

```
tasks.register<Jar>("fatJar") {
    archiveClassifier.set("all")
    duplicatesStrategy = DuplicatesStrategy.EXCLUDE
    manifest {
        attributes["Main-Class"] = "MainKt"
    }
    from(configurations.runtimeClasspath.get().map { if (it.isDirectory) it else zipTree(it) })
    with(tasks.jar.get() as CopySpec)
}

```

Step 2: Build the Fat JAR

```
./gradlew fatJar

```


Step 3: Run the Fat JAR

```
java -jar build/libs/MVNGRDKOTLINDEMO-1.0-SNAPSHOT-all.jar

```

Experiment - 5

Build and Run a Java Application with Maven, Migrate the Same Application to Gradle

🎯 Part 1: Create and Build a Java Application with Maven

🛠️ Step 1: Create a Maven Project in IntelliJ IDEA

1. **Open IntelliJ IDEA**
   - Launch **IntelliJ IDEA** and click on **File → New → Project**. 🌟
2. **Select Maven**
   - In the **New Project** window, choose **Maven** from the options on the left.
   - Check **Create from archetype** and select **maven-archetype-quickstart**.
   - Click **Next**. 👨‍💻
3. **Enter Project Details**
   - **GroupId**: `com.example`
   - **ArtifactId**: `MVNGRDLDEMO`
   - Click **Next** and then **Finish**. ✨
4. **Wait for IntelliJ to Load Dependencies**
   - IntelliJ will automatically download the Maven dependencies, so just relax for a moment. 😎

📝 Step 2: Update `pom.xml` to Add Build Plugin

To compile and package your project into a `.jar` file, you need to add the **Maven Compiler** and **Jar** plugins. 🔧

1. Open the `pom.xml` file. 📄
2. Add the following inside the `<project>` tag:

```
<build>
    <plugins>
        <!-- Compiler Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>

        <!-- Jar Plugin -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>3.2.0</version>
            <configuration>
                <archive>
                    <manifest>
                        <mainClass>com.example.App</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>

```

🔨 Step 3: Build and Run the Maven Project

1. **Open IntelliJ IDEA Terminal**

   Press **Alt + F12** to open the terminal. 🖥️

2. **Compile and Package the Project**

   Run the following commands to build the project:

```
mvn clean compile
mvn package

```


3. **Locate the JAR File**

   After running the above, your `.jar` file will be located at:

```
D:\Idea Projects\MVNGRDLDEMO\target\MVNGRDLDEMO-1.0-SNAPSHOT.jar

```


4. **Run the JAR File**

   To run the generated JAR file, use:

```
java -jar target\MVNGRDLDEMO-1.0-SNAPSHOT.jar

```

🚀 Part 2: Migrate Maven Project to Gradle

🔄 Step 1: Initialize Gradle in Your Project

1. **Open Terminal in IntelliJ IDEA**
   Make sure you're in the project directory:

```
cd "D:\Idea Projects\MVNGRDLDEMO"
```

2. **Run Gradle Init Command**
   Execute the following command to migrate your Maven project to Gradle:

```
gradle init --type pom

```

This command will convert your Maven `pom.xml` into a Gradle `build.gradle` file. 🏗️

🛠️ Step 2: Review and Update `build.gradle`

1. **Open** `build.gradle` in IntelliJ IDEA.
2. Ensure the following configurations are correct:

```
plugins {
    id 'java'
}

group = 'com.example'
version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    testImplementation 'junit:junit:4.13.2'
}

jar {
    manifest {
        attributes(
            'Main-Class': 'com.example.App'
        )
    }
}
```

🔨 Step 3: Build and Run the Gradle Project

1. **Clean and Build the Project**

   To clean and build your Gradle project, run:

```
gradle clean build

```

2. **Run the Generated JAR File**

   Now, run the generated JAR file using:

```
java -jar build/libs/MVNGRDLDEMO-1.0-SNAPSHOT.jar

```

Experiment - 6

Understanding and Working with Jenkins

Installation:

System Requirements
Memory 256 MB of RAM
Disk Space Depends on your projects
OS Windows, Mac, Ubuntu, Linux
Java 8 or 11 (JDK or JRE)
Installation on Windows

Watch This Video To Seamlessly Install Jenkins : Jenkins Installation - Step by Step Guide
Step 1 : Check Java is installed
Step 2 : Download Jenkins.war file
Step 3 : Goto cmd prompt and run command
java -jar jenkins.war --httpPort=8080
Step 4 : On browser goto http://localhost:8080
Step 5 : Provide admin password and complete the setup

Jenkins Configuration
How to change Home Directory
Step 1: Check your Jenkins Home > Manage Jenkins > Configure System
Step 2 : Create a new folder
Step 3 : Copy the data from old folder to new folder
Step 4 : Create/Update env variable JENKINS_HOME
Step 5 : Restart Jenkins
jenkins.xml
JENKINS_HOME

How to setup Git on Jenkins
Step 1 : Goto Manage Jenkins > Manage Plugins
Step 2 : Check if git is already installed in Installed tab
Step 3 : Else goto Available tab and search for git
Step 4 : Install Git
Step 5 : Check git option is present in Job Configuration

Create the first Job on Jenkins
How to connect to Git Remote Repository in Jenkins (GitHub)
Step 1 : Get the url of the remote repository
Step 2 : Add the git credentials on Jenkins
Step 3 : In the jobs configuration goto SCM and provide git repo url in git section
Step 4 : Add the credentials
Step 5 : Run job and check if the repository is cloned

How to use Command Line in Jenkins CLI
Faster, easier, integration
Step 1 : start Jenkins
Step 2 : goto Manage Jenkins - Configure Global Security - enable security
Step 3 : goto - http://localhost:8080/cli/
Step 4 : download jerkins-cli jar. Place at any location.
Step 5 : test the jenkins command line is working
java -jar jenkins-cli.jar -s http://localhost:8080 /help --username ＜userName＞ --password ＜password＞

How to create Users + Manage + Assign Roles
Step 1 : Create new users
Step 2 : Configure users
Step 3 : Create and manage user roles Role Based Authorization Strategy Plugin - download – restart jenkins
Step 4 : Manage Jenkins - Configure Global Security - Authorization - Role Based Strategy
Step 5 : Create Roles and Assign roles to users
Step 6 : Validate authorization and authentication are working properly

How to create jenkinsfile
Build ＞ Deploy ＞ Test ＞ Release
Jenkins file : Pipeline as a code
Step 1 : Start Jenkins
Step 2 : Install Pipeline Plugin
Step 3 : Create a new job
Step 4 : Create or get Jenkinsfile in Pipeline section
Step 5 : Run and check the output

Jenkins Pipeline
How to get jenkinsfile from Git SCM
Step 1 : Create a new job or use existing job (type : Pipeline)
Step 2 : Create a repository or GitHub
Step 3 : Add Jenkinsfile in the repo
Step 4 : Under Jenkins job ＞ Pipeline section ＞ Select Definition Pipeline script from SCM
Step 5 : Add repo and jenkinsfile location in the job under Pipeline section
Step 6 : Save & Run