Practical Maven

1) Prereqs (JDK, Maven, Git)

- **JDK**: Maven requires a JDK. Check `java -version` and set `JAVA_HOME`. (Use Java 11+ for general Maven; if you work on older frameworks like Spark 2.x, check their Java requirements — Spark 2 often expects Java 8).
  (Maven docs note JDK is required.) maven.apache.org
- **Install Maven**:
  - macOS (Homebrew):
    `brew update && brew install maven`
  - Ubuntu/Debian:
    `sudo apt update && sudo apt install maven`
  - Windows: download the binary ZIP from Apache Maven and add `mvn` to PATH.
    (Official install instructions.) maven.apache.orgHomebrew Formulae
- **Install Git**:
  - macOS: `brew install git`
  - Ubuntu: `sudo apt install git`
  - Windows: install from https://git-scm.com (or use Git for Windows)

Verify:

```
java -version
mvn -version
```

```
git --version
```

---

2) Create a *simple* Maven project (archetype quickstart)

Use Maven's quickstart archetype (creates `App.java` + `AppTest.java`) — exact command:

```
mvn archetype:generate \
  -DgroupId=com.example \
  -DartifactId=hello-maven \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DinteractiveMode=false
```

This creates the project folder `hello-maven/` with a `pom.xml`. (Archetype docs / quickstart.) maven.apache.org

Go into it:

```
cd hello-maven
tree   # or ls -R to view src/main/java and
src/test/java
```

---

3) Typical Maven lifecycle commands (clean, compile, test, package, install)

Run these from the project root (where `pom.xml` is):

```
mvn clean          # remove target/
mvn compile        # compile src/main/java
mvn test           # run unit tests (src/test)
mvn package        # build jar in target/
mvn install        # install to local ~/.m2/repository
```

`mvn package` produces `target/hello-maven-1.0-SNAPSHOT.jar` (or similar). (Maven "five minutes" / getting started covers these commands). maven.apache.org

To run the app (if it has a simple `main` class):

```
# option A: with exec plugin
mvn exec:java -Dexec.mainClass="com.example.App"

# option B: if jar is self-contained (no external deps)
java -cp target/hello-maven-1.0-SNAPSHOT.jar
com.example.App
```

If you have dependencies and want a single runnable (uber) jar, add `maven-shade-plugin` to `pom.xml` (see snippet below).

---

4) Example minimal `pom.xml` (quickstart + junit)

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>hello-maven</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.13.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Shade plugin (if you need a fat jar):

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>3.2.4</version>
      <executions>
        <execution>
          <phase>package</phase>
          <goals><goal>shade</goal></goals>
          <configuration>
            <transformers>
              <transformer
implementation="org.apache.maven.plugins.shade.resource.
ManifestResourceTransformer">
                <mainClass>com.example.App</mainClass>
              </transformer>
            </transformers>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

---

5) Git — create repo and push (quick steps)

Inside your project root:

```
echo "target/" > .gitignore
git init
git add .
git commit -m "Initial commit - maven quickstart"
# create remote on GitHub (manually or with gh CLI)
then:
git remote add origin
git@github.com:youruser/hello-maven.git
git branch -M main
git push -u origin main
```

If you want to create the repo from CLI: `gh repo create youruser/hello-maven --public --source=. --push`(GitHub CLI).

---

6) Common troubleshooting

- `Missing POM` error → you're not in the directory with `pom.xml` (run `ls` / `pwd`).
- `value toDF is not a member of Seq` (Scala/Spark) → `import spark.implicits._` (from earlier convo).

- Dependency not found → check `groupId:artifactId:version` is valid on Maven Central, remove accidental bogus dependencies (example `org.scala-lang:toolkit_2.12:0.5.0` is not a valid artifact for Scala 2.12).

---

7) Example GitHub repos you can clone & study

Simple (starter) — Java + Maven, tiny app with tests

`jenkins-docs/simple-java-maven-app` — used by Jenkins docs for the "Build a Java app with Maven" tutorial. Great for learning the basic lifecycle and tests. GitHub

Complex (Spark + Scala + Maven) — real-world example

`martinprobson/Spark-Scala-Maven-Example` — example Maven project showing how to build a Scala + Spark (Spark 2) application with a Maven `pom.xml` and packaging settings (useful if you're building Spark jobs with Maven). GitHub+1

(If you prefer a different "complex" sample, tell me whether you want **Spring Boot microservices**, **Spark streaming**, or **big data ETL** — I'll give a curated repo.)

Quick checklist you can copy

```
# prereqs
java -version
mvn -version
```

```
git --version

# create project
mvn archetype:generate -DgroupId=com.example
-DartifactId=hello-maven \
  -DarchetypeArtifactId=maven-archetype-quickstart
-DinteractiveMode=false

cd hello-maven
mvn clean compile test package
# run
mvn exec:java -Dexec.mainClass="com.example.App"
```