Scala + sbt project — step-by-step (terminal and IntelliJ)

Below is a compact, copy-pasteable guide you can follow start-to-finish.

It uses **Scala 3** syntax (but you can change `scalaVersion` if you prefer Scala 2).

I show a manual project, a `sbt new` template option, sample source + test, common sbt commands, and IntelliJ steps.

Quick prerequisites

- Java JDK 11+ (JDK 17 recommended) installed and `JAVA_HOME` set.
- sbt installed (or use the sbt launcher script).
- IntelliJ IDEA (Community or Ultimate) + **Scala plugin**.
- Terminal / shell access.

Project layout (what we'll create)

```
scala-sbt-sample/
├── build.sbt
├── project/
│   ├── build.properties
│   └── plugins.sbt
├── .gitignore
├── README.md
└── src/
    ├── main/
    │   └── scala/
    │       └── example/
    │           ├── Hello.scala
    │           └── Greeter.scala
    └── test/
        └── scala/
            └── example/
                └── HelloSuite.scala
```

Files — copy/paste ready

`project/build.properties`

```
sbt.version=1.8.2
```

(Change sbt.version to your installed sbt if needed.)

`project/plugins.sbt`

(keep empty or add plugins like sbt-assembly / scalafmt)

```
// Example: add sbt-assembly if you need a fat jar
// addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "1.2.0")
```

build.sbt

```
ThisBuild / scalaVersion := "3.2.2"
ThisBuild / version := "0.1.0-SNAPSHOT"

lazy val root = (project in file("."))
  .settings(
    name := "scala-sbt-sample",
    libraryDependencies ++= Seq(
      // Test framework (optional) - replace with latest version if
needed
      "org.scalatest" %% "scalatest" % "3.2.16" % Test
    )
  )
```

Tip: Replace `"3.2.2"` and library versions with your preferred/latest versions.

src/main/scala/example/Hello.scala

```
package example

@main def hello(name: String = "World"): Unit =
  println(s"Hello, $name! Running Scala
${util.Properties.versionNumberString}")
```

src/main/scala/example/Greeter.scala

```
package example

object Greeter:
  def greet(name: String): String = s"Hello, $name"
```

src/test/scala/example/HelloSuite.scala

```
package example

import org.scalatest.funsuite.AnyFunSuite

class HelloSuite extends AnyFunSuite:
  test("Greeter.greet should include the name") {
    assert(Greeter.greet("Dani").contains("Dani"))
  }
```

`.gitignore` (minimal)

```
target/
project/target/
.idea/
*.iml
.cache
.DS_Store
```

`README.md` (starter)

```
# scala-sbt-sample

Simple Scala 3 + sbt sample project.

## Build & Run
- `sbt compile`
- `sbt run`
- `sbt test`
```

Create the project from terminal (two ways)

A. Quick — use sbt template (fast)

```
# interactive template, will ask project name etc.
sbt new scala/scala3.g8
# then:
cd <your-project-dir>
```

```
sbt
# inside sbt shell:
# compile, run, test
compile
run
test
```

B. Manual — exact commands (copy/paste)

```
mkdir scala-sbt-sample
cd scala-sbt-sample
mkdir -p src/main/scala/example src/test/scala/example project

# create files (using heredoc)
cat > project/build.properties <<'EOF'
sbt.version=1.8.2
EOF

cat > build.sbt <<'EOF'
ThisBuild / scalaVersion := "3.2.2"
ThisBuild / version := "0.1.0-SNAPSHOT"

lazy val root = (project in file("."))
  .settings(
    name := "scala-sbt-sample",
    libraryDependencies ++= Seq(
      "org.scalatest" %% "scalatest" % "3.2.16" % Test
    )
  )
EOF

cat > src/main/scala/example/Hello.scala <<'EOF'
package example

@main def hello(name: String = "World"): Unit =
  println(s"Hello, $name! Running Scala
${util.Properties.versionNumberString}")
EOF
```

```
cat > src/main/scala/example/Greeter.scala <<'EOF'
package example

object Greeter:
  def greet(name: String): String = s"Hello, $name"
EOF

cat > src/test/scala/example/HelloSuite.scala <<'EOF'
package example

import org.scalatest.funsuite.AnyFunSuite

class HelloSuite extends AnyFunSuite:
  test("Greeter.greet should include the name") {
    assert(Greeter.greet("Dani").contains("Dani"))
  }
EOF

echo "target/" > .gitignore
echo "# scala-sbt-sample" > README.md
```

Now run:

```
sbt compile
sbt run          # will run the main; pick the @main method
automatically
sbt "runMain example.Hello"  # alternative
sbt test
sbt console      # opens Scala REPL with your project classpath
```

IntelliJ IDEA — create or import (step-by-step)

Option 1 — Create new sbt project from IntelliJ

1. File → New → Project.
2. Choose **Scala** (sbt), Project SDK → select JDK (17 recommended).
3. Build tool: **sbt**. Choose Scala version (e.g., 3.2.2).
4. Finish — IntelliJ creates `build.sbt` etc.
5. Create the `src/main/scala/...` and `src/test/scala/...` packages and paste the sample files above.
6. IntelliJ will auto-import sbt build; wait for indexing.

Option 2 — Import existing sbt project

1. File → New → Project from Existing Sources.
2. Select the folder with `build.sbt`.
3. IntelliJ detects sbt; choose to import as sbt project.
4. Wait for sbt to sync / download dependencies.

Run / Test inside IntelliJ

- Open `Hello.scala` — click the green run icon near the `@main` function to run.
- Or create a Run Configuration: Run → Edit Configurations → **Application** → Main class `example.Hello` (or for Scala 3 `example.Hello`), use sbt shell for run if preferred.
- Run tests: right-click test class → Run, or use the test tool window.
- Use the **sbt shell** tab (bottom) to run `compile`, `run`, `test` exactly as in terminal.

Useful sbt commands (cheat sheet)

- `sbt compile` — compile sources
- `sbt run` — run main (if multiple mains it will prompt)
- `sbt "runMain example.Hello arg1 arg2"` — run specific main
- `sbt test` — run tests
- `sbt console` — Scala REPL with project classpath
- `sbt clean` — remove build artifacts
- `sbt package` — produce a JAR under `target/`
- `sbt assembly` — produce fat JAR (if sbt-assembly plugin is added)
- `sbt publishLocal` — publish artifact to local ivy repo

Tips & recommendations

- Keep `scalaVersion` in `ThisBuild` so all modules (if you add more) use the same Scala.
- Use `sbt new scala/scala3.g8` for quick prototype.
- Add `scalafmt` (formatter) and `scalafix` for code quality.
- For production fat JARs add `sbt-assembly` or use `sbt-native-packager`.
- For multi-module projects, split modules with `lazy val core = (project in file("core")).settings(...)`.

Small multi-module example (in `build.sbt`)

```
lazy val common = (project in file("common")).settings(
```

```
  name := "common"
)

lazy val app = (project in file("app"))
  .dependsOn(common)
  .settings(
    name := "app"
  )
```