**EBook Gratis**

# APRENDIZAJE sbt

Free unaffiliated eBook created from
**Stack Overflow contributors.**

#sbt

# Tabla de contenido

# Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sbt

It is an unofficial and free sbt ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sbt.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Capítulo 1: Empezando con sbt

## Observaciones

La herramienta de compilación simple (SBT, por sus siglas en inglés) se puede usar para compilar el código del proyecto Scala (o Java). Esto incluye el código de administración, las dependencias y los recursos que se deben construir, probar y / o compilar en un `.jar` u otro artefacto. Se pueden crear tareas personalizadas para gestionar todos estos procesos.

Una nota sobre el nombre; SBT a veces se denomina 'Herramienta de compilación de Scala'. Si bien esta no fue la intención original, también se ha utilizado comúnmente. SBT se puede usar para construir cualquier proyecto en la JVM.

`.sbt` archivos `.sbt` o 'definiciones de compilación SBT' son archivos especialmente interpretados, escritos en Scala, que SBT utiliza para definir una compilación. `.scala` definiciones de compilación `.scala` también se pueden escribir e importar en un archivo `.sbt` .

Las versiones anteriores a `13.6` requerían que cualquier archivo `.sbt` tenga cada instrucción separada por una línea en blanco. Sin la línea en blanco, el archivo `.sbt` se romperá.

Existe un paquete universal en formatos ZIP y TGZ .

## Versiones

| Versión | Estado | Fecha de lanzamiento |
|---|---|---|
| 0.13.12 | Estable | 2016-07-17 |

## Examples

### Instalar SBT en Linux

Las instrucciones completas se pueden encontrar aquí .

1. Instale el JDK .
2. Establecer la variable de entorno de Java.

```
export JAVA_HOME=/usr/local/java/jdk1.8.0_102
echo $JAVA_HOME
/usr/local/java/jdk1.8.0_102
export PATH=$PATH:$JAVA_HOME/bin/
echo $PATH
...:/usr/local/java/jdk1.8.0_102/bin/
```

3. Instala Scala.

```
sudo wget http://www.scala-lang.org/files/archive/scala-2.11.8.deb
sudo dpkg -i scala-2.11.8.deb
sudo apt-get update
sudo apt-get install scala
```

4. Instale SBT.

```
wget https://bintray.com/artifact/download/sbt/debian/sbt-0.13.9.deb
sudo dpkg -i sbt-0.13.9.deb
sudo apt-get update
sudo apt-get install sbt
```

# Distribuciones Linux basadas en RPM

- Descargue las definiciones del repositorio SBT y agréguelo a YUM:

```
curl https://bintray.com/sbt/rpm/rpm | sudo tee /etc/yum.repos.d/bintray-sbt-rpm.repo
```

- Instale SBT de acuerdo con las definiciones agregadas previamente a YUM:

```
sudo yum install sbt
```

### Instalar SBT en Windows

# Instalar

Los instaladores de MSI se pueden encontrar aquí . Esta es la última versión estable . Descargar y ejecutar para instalar.

# Verificar la instalacion

- Utilice el `WindowsKey + R` , escriba `cmd` .
- Alternativamente, navegue a `.sbt` (por ejemplo, en `C:\Users\Hopper` ) y escriba `cmd` en la barra de direcciones.
- Escriba `sbt about` para obtener información `sbt about` versión, verificando que esté instalada. Debería ver algo como esto:

```
Java HotSpot(TM) 64-But Server VM warning: ignoring option MaxPermSize=256m; support was
removed in 8.0
[info] Set current project to root--sbt (in build file:/C:/Users/Hopper/.sbt/)
[info] This is sbt 0.13.8
...
```

### Instalar en Mac OSX

Las instrucciones oficiales completas se pueden encontrar aquí .

# MacPorts

Instalar MacPorts . Luego, en el terminal ejecute:

```
port install sbt
```

# Homebrew

Instale Homebrew . Luego, en el terminal ejecute:

```
brew install sbt
```

# Fuentes

Descargue la instalación de *todas las plataformas* (tgz) de sbt desde SBT .

```
sudo su
cd /opt
mkdir sbt
cd sbt
curl https://dl.bintray.com/sbt/native-packages/sbt/0.13.13/sbt-0.13.13.tgz -o sbt-0.13.13.tgz
```

Luego, ejecuta siguiendo

```
tar zxf sbt-0.13.13.tgz
ln -s sbt-0.13.13 latest
```

Dentro de $ HOME asegúrese de actualizar ~ / .profile - agregando las siguientes líneas

```
export SBT_HOME=/opt/sbt/latest
export PATH=$PATH:$SBT_HOME/bin
```

# Verificación

En la terminal ejecute:

```
which sbt
```

Usted debe esperar una salida similar a:

```
/opt/local/bin/sbt
```

Si no obtienes ningún resultado, sbt no está instalado.

## Importar proyecto SBT en Eclipse

Esto supone que ha instalado tanto Eclipse como SBT .

- Instale el complemento SBT para Eclipse desde el mercado de Eclipse.
- En la línea de comandos, cambie el directorio al directorio raíz del proyecto.

  ```
  $ cd ~/home/sample/project
  ```

- Ejecuta sbt, que cargará el proyecto.

  ```
  $ sbt
  ```

- Compilar el proyecto para asegurar que las dependencias son obtenibles.

  ```
  > compile
  ```

- Ejecutar la tarea de `eclipse` :

  ```
  > eclipse
  ```

- Entra en Eclipse y selecciona la opción de menú:

  ```
  File > New > Project From Existing Sources
  ```

- En el asistente, navegue hasta el directorio de su proyecto y selecciónelo. Eclipse se encargará del resto.

Lea Empezando con sbt en línea: https://riptutorial.com/es/sbt/topic/2351/empezando-con-sbt

# Capítulo 2: Dependencias

## Examples

### Agregar una dependencia de biblioteca administrada

`libraryDependency` es el `SettingKey` que maneja las dependencias de la biblioteca 'administrada', que son dependencias que se descargan automáticamente y coinciden con las versiones suministradas. Para agregar una sola dependencia:

```
libraryDependencies += "com.typesafe.slick" %% "slick" % "3.2.0-M1"
```

La primera parte, `"com.typesafe.slick"` , indica el paquete de la biblioteca. La segunda parte, `"slick"` , es la biblioteca en cuestión. La parte final, `"3.2.0-M1"` , es la versión. Debido a que la biblioteca está unida por `%%` , se utilizará la versión de Scala proporcionada por la clave de configuración de `scalaVersion` .

Puedes agregar múltiples bibliotecas a la vez usando `++=` :

```
libraryDependencies ++= Seq(
  "com.typesafe.slick" %% "slick" % "3.2.0-M1" % "compile",
  "com.typesafe.slick" %% "slick-hikaricp" % "3.2.0-M1",
  "mysql" % "mysql-connector-java" % "latest.release"
)
```

Recuerde la naturaleza funcional de Scala, lo que le permite calcular las dependencias. Solo recuerda devolver una `Seq` :

```
libraryDependencies ++= {
  lazy val liftVersion = "3.0-RC3" //Version of a library being used
  lazy val liftEdition = liftVersion.substring(0,3) //Compute a value
  Seq(
    "net.liftweb" %% "lift-webkit" % liftVersion % "compile",  // Use var in Seq
    "net.liftmodules" %% ("ng_" + liftEdition) % "0.9.2" % "compile",  // Use computed var in
Seq
  )  // Because this is the last statement, the Seq is returned and appended to
libraryDependencies
}
```

### Añadir un repositorio

Un repositorio es un lugar donde SBT busca `libraryDependencies` de `libraryDependencies` . Si la compilación se queja de no encontrar una dependencia, puede faltar el repositorio correcto. Dentro de SBT, los repositorios se enumeran en los `resolvers` SettingKey:

```
resolvers += "Flyway" at "https://flywaydb.org/repo"
```

Esto sigue la sintaxis de 'Nombre del repositorio' en 'ubicación url'.

## Biblioteca de pines a la versión del proyecto de Scala

Si tu proyecto tiene esto:

```
scalaVersion := 2.11  // Replace '2.11' with the version of Scala your project is running on
```

Luego, puede usar `%%` para obtener automáticamente la versión de la biblioteca compilada contra la versión de Scala que está utilizando el proyecto:

```
libraryDependencies += "com.typesafe.slick" %% "slick" % "3.2.0-M1"
```

Tenga en cuenta que tener las dos líneas anteriores es equivalente a tener esta línea:

```
libraryDependencies += "com.typesafe.slick" % "slick_2.11" % "3.2.0-M1"
```

## Biblioteca de pines a la versión específica de Scala

Una biblioteca se puede "anclar" a una versión específica de Scala usando el operador `%` entre `groupId` y `artifactId` (las dos primeras cadenas en una dependencia de biblioteca). En este ejemplo, fijamos la biblioteca con el `artifactId` de `slick` a la versión `2.10` Scala:

```
 libraryDependencies += "com.typesafe.slick" % "slick_2.10" % "3.2.0-M1"
```

Lea Dependencias en línea: https://riptutorial.com/es/sbt/topic/6760/dependencias

# Capítulo 3: Descripción general de la construcción

## Observaciones

La documentación oficial está en www.scala-sbt.org .

## Examples

### Estructura de directorios

La estructura estándar para un proyecto construido por SBT es:

```
projectName/
    build.sbt
    project/
      <SBT sub-build information>
    src/
      main/
        scala/
           <Scala source files>
        java/
           <Java source files>
        resources/
           <Resource files>
      test/
        scala/
           <Scala test files>
        java/
           <Java test files>
        resources/
           <Resource files>
```

Pueden existir otros directorios, pero la construcción trata principalmente con estos. En el directorio base se coloca `build.sbt` , cuyos contenidos como mínimo son:

- `name := <name of build>` : este es el nombre del proyecto.
- `version := <version number>` : Esta es la versión del proyecto para referenciar en el código descendente.
- `scalaVersion := <version of Scala>` : esta es la versión de Scala contra la que se construye el bytecode del proyecto.

El directorio del `project` es donde se colocan los archivos de `meta-build` (en oposición a la `proper-build` ). Este directorio puede tener su propio archivo `build.sbt` que se ejecuta exactamente de la misma manera, creando un entorno para que se ejecute la `proper-build` SBT `proper-build` . Esto es recursivo, por lo que el directorio del `project` puede tener su propio directorio del `project` donde se produce una `meta-meta-build` , y así sucesivamente.

Al construir, SBT creará un directorio de `target` en el que se colocan los archivos de clase y otros componentes.

### Hoja de trucos

Esta hoja asume que usted está en el directorio raíz del proyecto, que contiene el `build.sbt` . `$` indica un indicador de comando y `>` indica que los comandos se ejecutan dentro de la consola SBT.

## Compilar un proyecto

```
$ sbt compile
```

## Probar un proyecto

```
$ sbt test
```

## Introduzca SBT REPL:

```
$ sbt
```

## Entrar en la Consola Scala con Proyecto Construido Disponible

```
$ sbt
> console
```

## Generar Scaladoc

Este es un ejemplo de ejecución de una 'tarea' de SBT . El sitio SBT tiene más información sobre la generación de documentación Scaladoc .

```
$ sbt doc
```

o:

```
$ sbt
> doc
```

Lea Descripción general de la construcción en línea:
https://riptutorial.com/es/sbt/topic/6761/descripcion-general-de-la-construccion

# Capítulo 4: Empezando con el desarrollo diario.

## Examples

### Ejemplo de desarrollo continuo diario con Scala.

```
# install sbt with homebrew (if you didn't)
brew install sbt

# - create a new scala project
# - name your project name when asked like: hello-world
sbt new sbt/scala-seed.g8

# go to the new project directory that you named
cd hello-world

# run sbt to open the sbt shell
sbt

# run your project in continuous running mode
~ run

# to continuously see the test outputs
# open up a new terminal tab, run sbt, type:
~ test

# to continuously compile
# open up a new terminal tab, run sbt, type:
~ compile
```

**~** utilizado para operaciones continuas en sbt como se ve arriba.

Lea Empezando con el desarrollo diario. en línea:
https://riptutorial.com/es/sbt/topic/9842/empezando-con-el-desarrollo-diario-

# Capítulo 5: Proyectos

## Examples

### Múltiples proyectos en la misma compilación (subproyectos)

A veces, una compilación combina varios directorios de origen, cada uno de los cuales es su propio 'proyecto'. Por ejemplo, podría tener una estructura de construcción como esta:

projectName / build.sbt project / src / main / ... test / ... core / src / main / ... test / ... webapp / src / main / ... test / ...

En el proyecto anterior, el código en `projectName/src` se considera el proyecto `root` . Hay otros dos módulos, o 'subproyectos', `core` y `webapp` .

La configuración de un subproyecto es similar a la configuración del proyecto raíz, excepto que el subdirectorio se especifica en el proyecto. Este ejemplo muestra un proyecto raíz que agrega un proyecto `core` y una aplicación `webapp` .

```
lazy val root = (project in file(".")).aggregate(core,webapp).dependsOn(core, webapp)

lazy val core = (project in file("core"))

lazy val webapp = (project in file("webapp")).dependsOn(core)
```

Los valores pasados a `file()` son los directorios relativos a la raíz del proyecto.

El proyecto de la aplicación `webapp` depende del proyecto `core` , que se indica en la cláusula `dependsOn` la `dependsOn` , que toma el valor `core` especificado en la línea anterior. `dependsOn` y la evaluación `lazy` asegura que las dependencias estén disponibles antes de que los proyectos las utilicen. En este caso, la aplicación `webapp` depende del `core` , por lo que el `core` se compilará antes de que la compilación intente compilar la aplicación `webapp` .

`aggregate` hace que las tareas definidas en un proyecto estén disponibles para el proyecto que lo agregue. Por ejemplo, la ejecución de `compile` en el proyecto `root` también ejecutará la `compile` en `core` y `webapp` .

### Configurar macros en un proyecto

En el archivo `build.sbt` (o donde se define el proyecto si está en otra ubicación), agregue la siguiente configuración:

```
scalacOptions += "-language:experimental.macros"
```

Por ejemplo, un proyecto podría definirse así:

```
lazy val main = project.in(file("."))  // root project
```

```
  .settings(scalacOptions += "-language:experimental.macros",
            addCompilerPlugin("org.scalamacros" % "paradise" % "2.1.0" cross
CrossVersion.full))
```

En el ejemplo anterior, el `paradise` plugin se incluyó a fin de proporcionar soporte completo de Scala `2.10.x` .

## Configuraciones de pantalla

Cuando esté en la consola SBT, para enumerar todas las configuraciones definibles para un proyecto:

```
settings
```

O, para obtener la configuración de un subproyecto (por ejemplo, llamada `webapp` ):

```
project webapp
settings
```

La primera línea de arriba se desplaza al subproyecto específico.

Para mostrar el valor de una configuración específica (por ejemplo, `organization` ):

```
show organization
```

Esto mostrará el valor de esa configuración.

Lea Proyectos en línea: https://riptutorial.com/es/sbt/topic/6790/proyectos

# Capítulo 6: Tareas

## Examples

### Crear una tarea simple

Todo lo que se necesita para definir una tarea es una declaración de su tipo y una descripción:

```
lazy val exampleTask = taskKey[Unit]("An example task that will return no value.")
```

Debido a que `Unit` es el tipo, esta tarea está compuesta completamente de efectos secundarios. Una vez definido, para implementar acciones:

```
exampleTask := {
  val s: TaskStreams = streams.value
  s.log.info("The example task was executed.")
}
```

Si estos están definidos en `build.sbt` , puede cargar el proyecto y ejecutarlo:

```
> exampleTask
[info] The example task was executed.
```

Lea Tareas en línea: https://riptutorial.com/es/sbt/topic/7542/tareas

# Creditos

| S. No | Capítulos | Contributors |
|---|---|---|
| 1 | Empezando con sbt | Altius, andriosr, Ani Menon, Community, Eugene Yokota, James , karel, kn_pavan, mko, Nathaniel Ford |
| 2 | Dependencias | Nathaniel Ford |
| 3 | Descripción general de la construcción | Nathaniel Ford |
| 4 | Empezando con el desarrollo diario. | Inanc Gumus |
| 5 | Proyectos | Nathaniel Ford |
| 6 | Tareas | Nathaniel Ford |