sbt

simple build tool

sbt

~~simple~~ build tool
scala

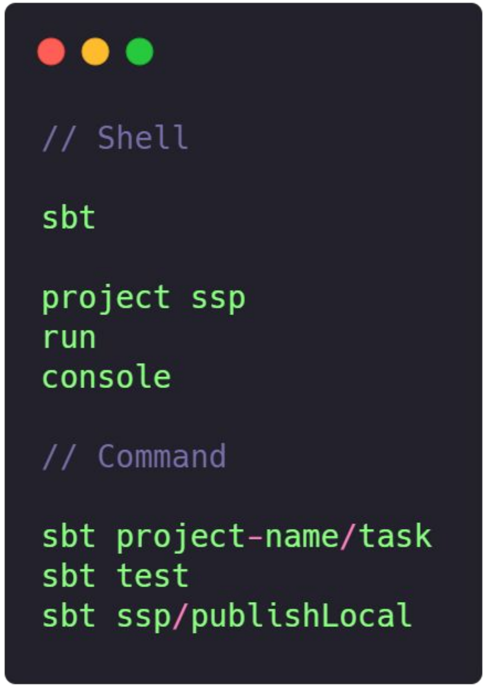
Principles

Automate the creation of an executable for your application

- Manage dependencies
- Decide what need to be built, in which order and using which code
- Use caching to speed up tasks
- Create custom plugins to automate even more things

Shell

- Run as an interactive shell or as bash commands
- Autocomplete, history and tips
- Specify a scope or run tasks globally



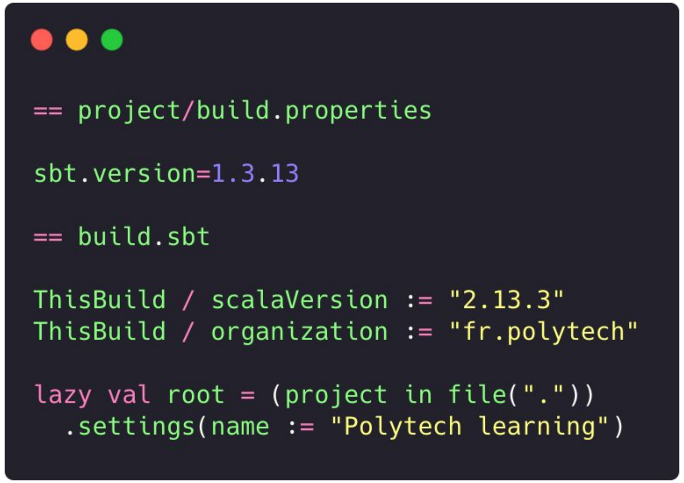
```
// Shell

sbt

project ssp
run
console

// Command

sbt project-name/task
sbt test
sbt ssp/publishLocal
```

Bootstrap

- **Always specify your project versions**
  To make sure anyone uses the same

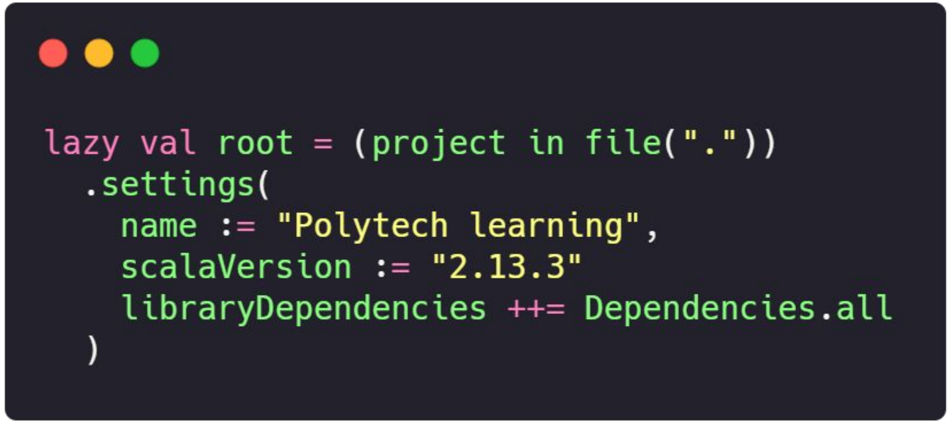
- **Need to install sbt locally**
  Or good luck without it ;)
  https://www.scala-sbt.org/1.x/docs/Setup.html

- **Sbt will create `target` directories**
  Used for caching, can clean with with `sbt clean`



```
== project/build.properties

sbt.version=1.3.13

== build.sbt

ThisBuild / scalaVersion := "2.13.3"
ThisBuild / organization := "fr.polytech"

lazy val root = (project in file("."))
  .settings(name := "Polytech learning")
```

Build definition



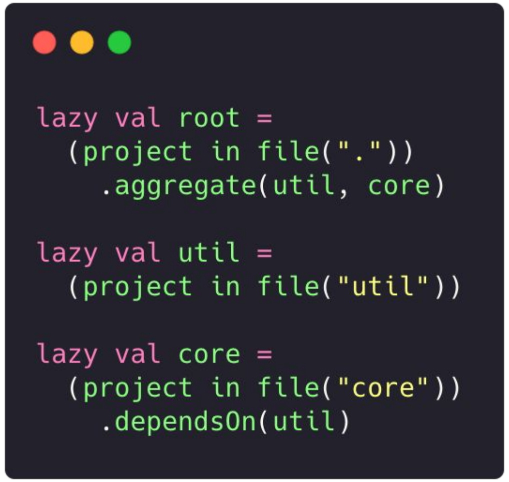
```scala
lazy val root = (project in file("."))
  .settings(
    name := "Polytech learning",
    scalaVersion := "2.13.3"
    libraryDependencies ++= Dependencies.all
  )
```

- Scala syntax
- Define a project and its settings
- Manage multiple modules with their own settings & dependencies

Projects

- Aggregate projects
  tasks will apply to all submodules

- Depends on another project
  will pull its dependencies

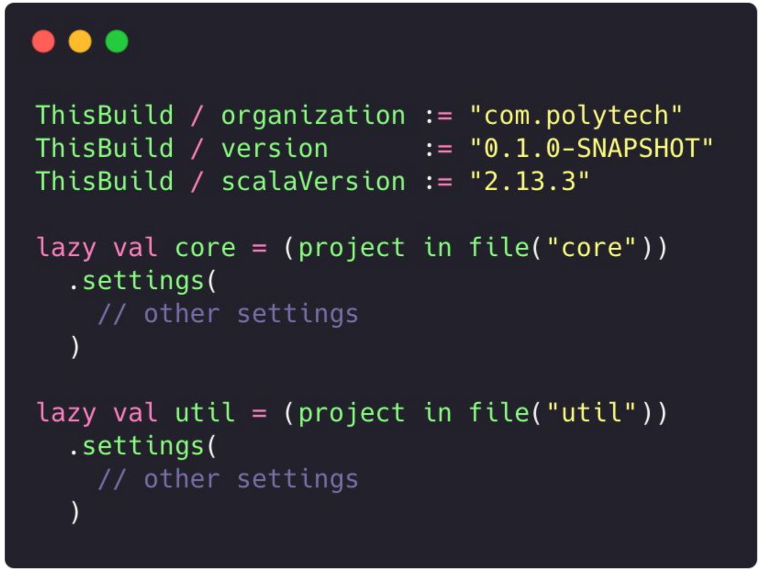


```scala
lazy val root =
  (project in file("."))
    .aggregate(util, core)

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

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

Build-wide settings

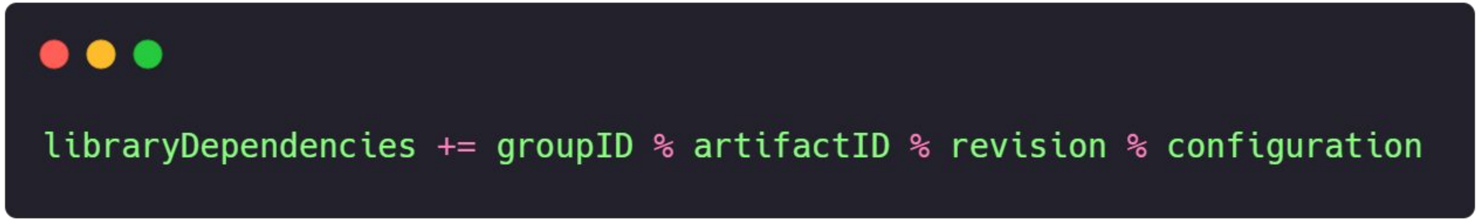
- Will be defined for all projects

- Overridable
  by the settings of each project



```
ThisBuild / organization := "com.polytech"
ThisBuild / version      := "0.1.0-SNAPSHOT"
ThisBuild / scalaVersion := "2.13.3"

lazy val core = (project in file("core"))
  .settings(
    // other settings
  )

lazy val util = (project in file("util"))
  .settings(
    // other settings
  )
```

Dependencies



```
libraryDependencies += groupID % artifactID % revision % configuration
```

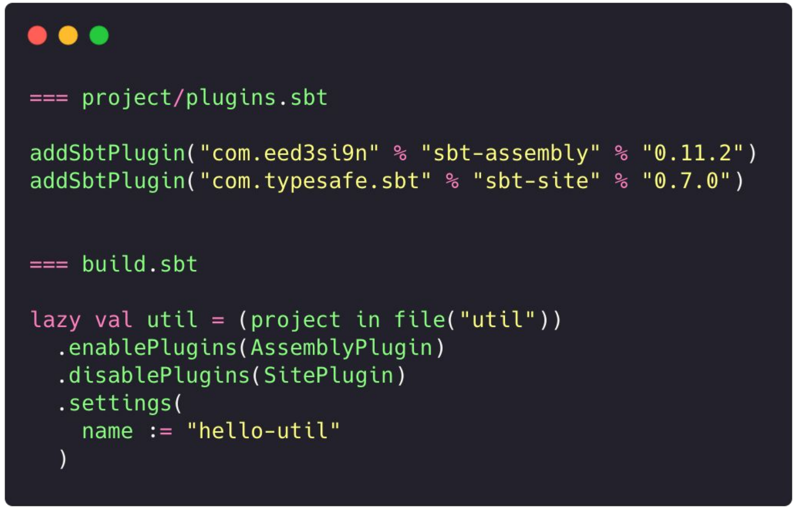
- Add libraries to your classpath
- Manage and add resolvers to download libraries
- Define a scope on which to provide the deps

Plugins

- Auto-plugin
  Since 0.13.5 version, no need to enable them manually

- Display all plugins of the project
  sbt plugins

- Global plugins
  ~/.sbt/<sbt_version>/plugins/plugins.sbt



```
=== project/plugins.sbt

addSbtPlugin("com.eed3si9n" % "sbt-assembly" % "0.11.2")
addSbtPlugin("com.typesafe.sbt" % "sbt-site" % "0.7.0")


=== build.sbt

lazy val util = (project in file("util"))
  .enablePlugins(AssemblyPlugin)
  .disablePlugins(SitePlugin)
  .settings(
    name := "hello-util"
  )
```

Plugins

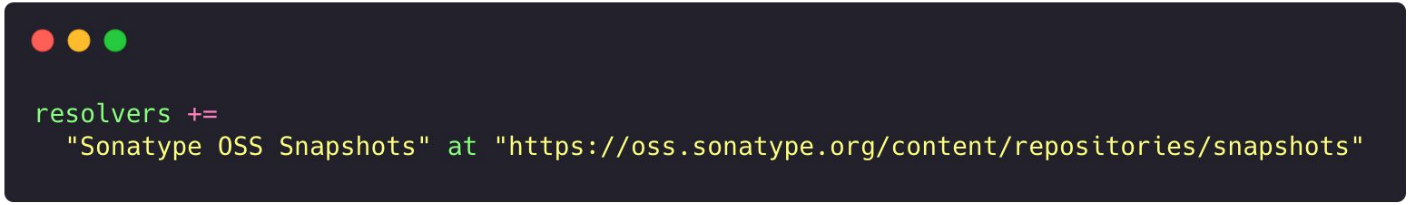
- https://github.com/sbt/sbt-release

- https://github.com/sbt/sbt-assembly
- https://github.com/marcuslonnberg/sbt-docker

- https://github.com/rtimush/sbt-updates

- https://github.com/sbt/sbt-groll

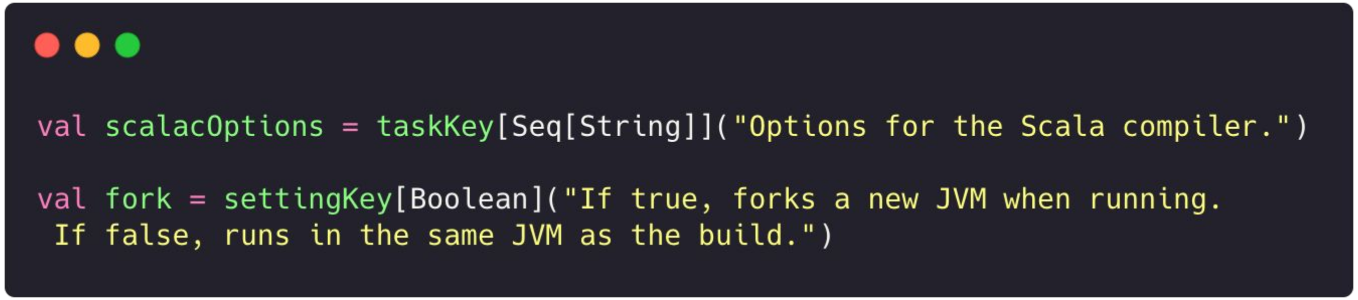
Resolvers



```
resolvers +=
  "Sonatype OSS Snapshots" at "https://oss.sonatype.org/content/repositories/snapshots"
```

- Add it to a project, usually at the root to provide it to all sub modules
- May be needed for some plugins, in this case add it to `projects/plugins.sbt`

Task vs Setting



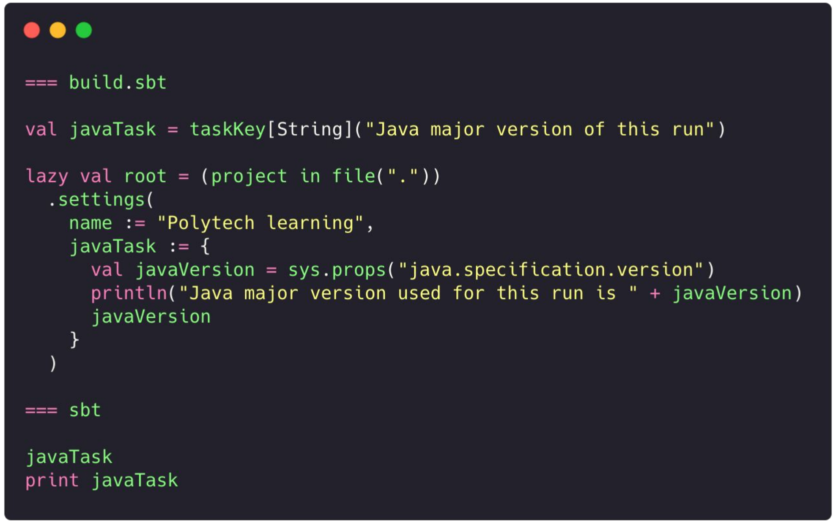
```scala
val scalacOptions = taskKey[Seq[String]]("Options for the Scala compiler.")

val fork = settingKey[Boolean]("If true, forks a new JVM when running.
 If false, runs in the same JVM as the build.")
```

- Tasks are evaluated every time they are called
- Settings are evaluated only once at load time

How to define a custom tasks

- The definition is linked to the project

- Available in the shell
  Remember that each call evaluate the content again



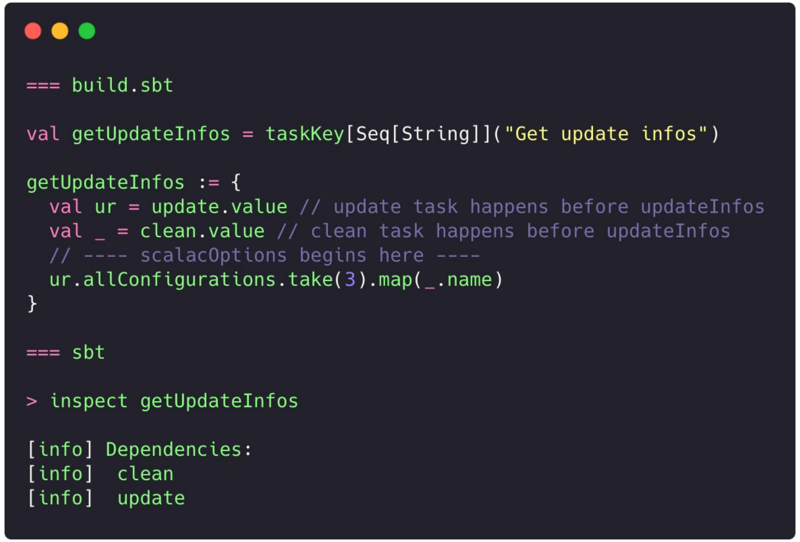
```
=== build.sbt

val javaTask = taskKey[String]("Java major version of this run")

lazy val root = (project in file("."))
  .settings(
    name := "Polytech learning",
    javaTask := {
      val javaVersion = sys.props("java.specification.version")
      println("Java major version used for this run is " + javaVersion)
      javaVersion
    }
  )

=== sbt

javaTask
print javaTask
```

Task graph

- Tasks can depend on other tasks
  They will be evaluated beforehand

- Tasks can depend on settings

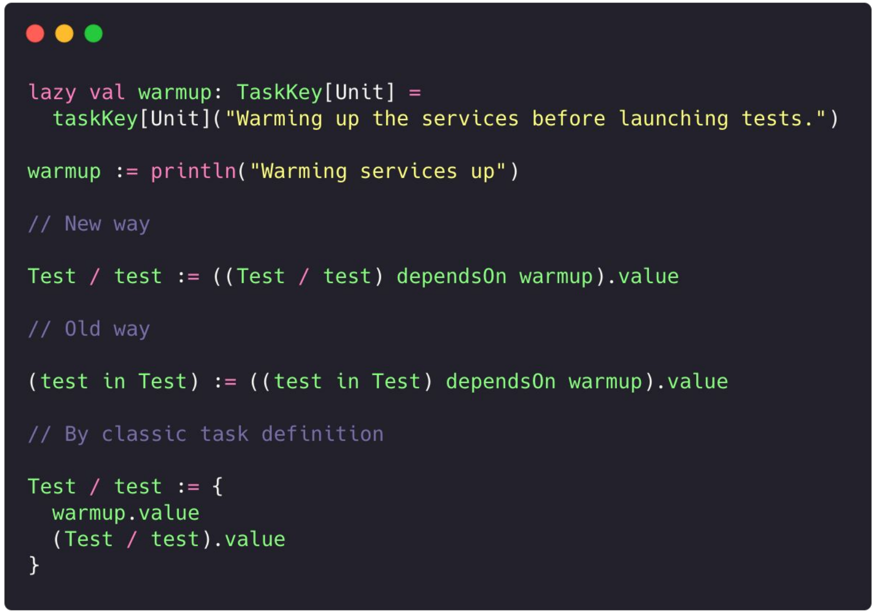
- Settings cannot depend on tasks
  As settings are loaded only once at boot time



```
=== build.sbt

val getUpdateInfos = taskKey[Seq[String]]("Get update infos")

getUpdateInfos := {
  val ur = update.value // update task happens before updateInfos
  val _ = clean.value // clean task happens before updateInfos
  // ---- scalacOptions begins here ----
  ur.allConfigurations.take(3).map(_.name)
}

=== sbt

> inspect getUpdateInfos

[info] Dependencies:
[info]   clean
[info]   update
```

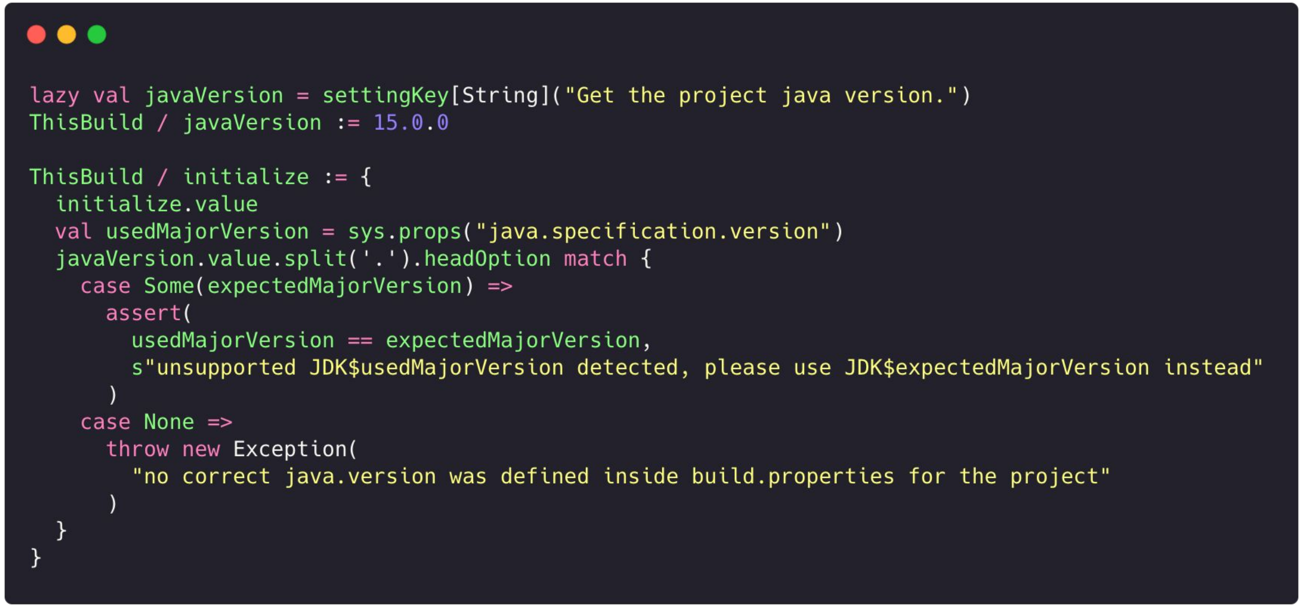
Task dependsOn

Make it clearer when you want to make tasks depend on others



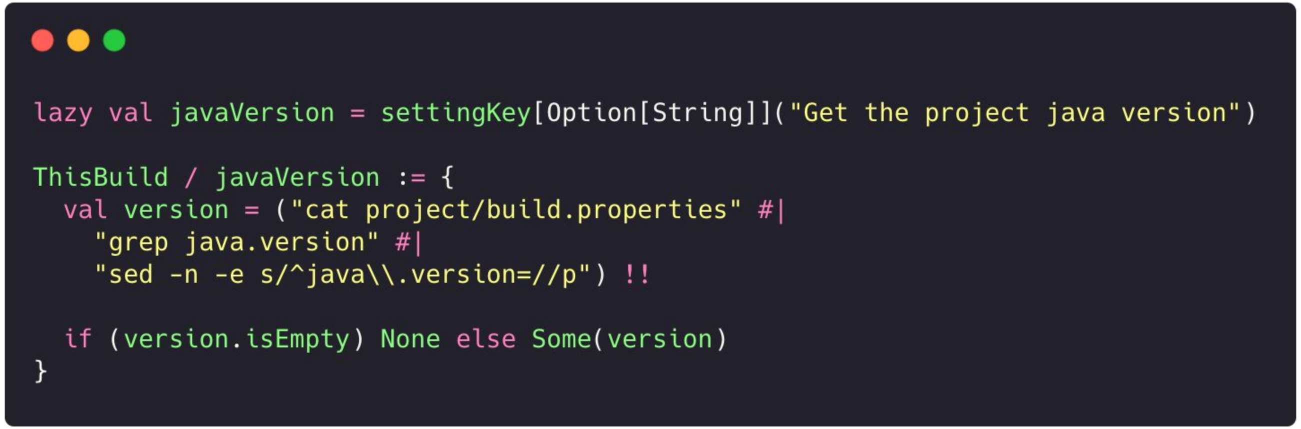
```scala
lazy val warmup: TaskKey[Unit] =
  taskKey[Unit]("Warming up the services before launching tests.")

warmup := println("Warming services up")

// New way

Test / test := ((Test / test) dependsOn warmup).value

// Old way

(test in Test) := ((test in Test) dependsOn warmup).value

// By classic task definition

Test / test := {
  warmup.value
  (Test / test).value
}
```

Take advantage of an existing task



```scala
lazy val javaVersion = settingKey[String]("Get the project java version.")
ThisBuild / javaVersion := 15.0.0

ThisBuild / initialize := {
  initialize.value
  val usedMajorVersion = sys.props("java.specification.version")
  javaVersion.value.split('.').headOption match {
    case Some(expectedMajorVersion) =>
      assert(
        usedMajorVersion == expectedMajorVersion,
        s"unsupported JDK$usedMajorVersion detected, please use JDK$expectedMajorVersion instead"
      )
    case None =>
      throw new Exception(
        "no correct java.version was defined inside build.properties for the project"
      )
  }
}
```

Go further and read from a file



```
lazy val javaVersion = settingKey[Option[String]]("Get the project java version")

ThisBuild / javaVersion := {
  val version = ("cat project/build.properties" #|
    "grep java.version" #|
    "sed -n -e s/^java\\.version=//p") !!

  if (version.isEmpty) None else Some(version)
}
```

Because who doesn’t like some bash and regex

History / versions

sbt 0.13.x	August 2013… still used in production
sbt 1.0.x	August 2017
sbt 1.1.x	January 2018
sbt 1.2.x	July 2018
sbt 1.3.x	September 2019
sbt 1.4.x	Incoming (RC1 September 2020)

Disadvantages

- Hard to master
  But the necessary basics are not

- Hard to bootstrap on your own
  But becoming more and more easier

- Caching is “bad” because too many things are free to change
  sbt cannot determine a task result in advance, which is why the build tool is “slow”

- Starting to get challenged
  And not many contributors to widen the gap with the competition

Alternatives

Maven™

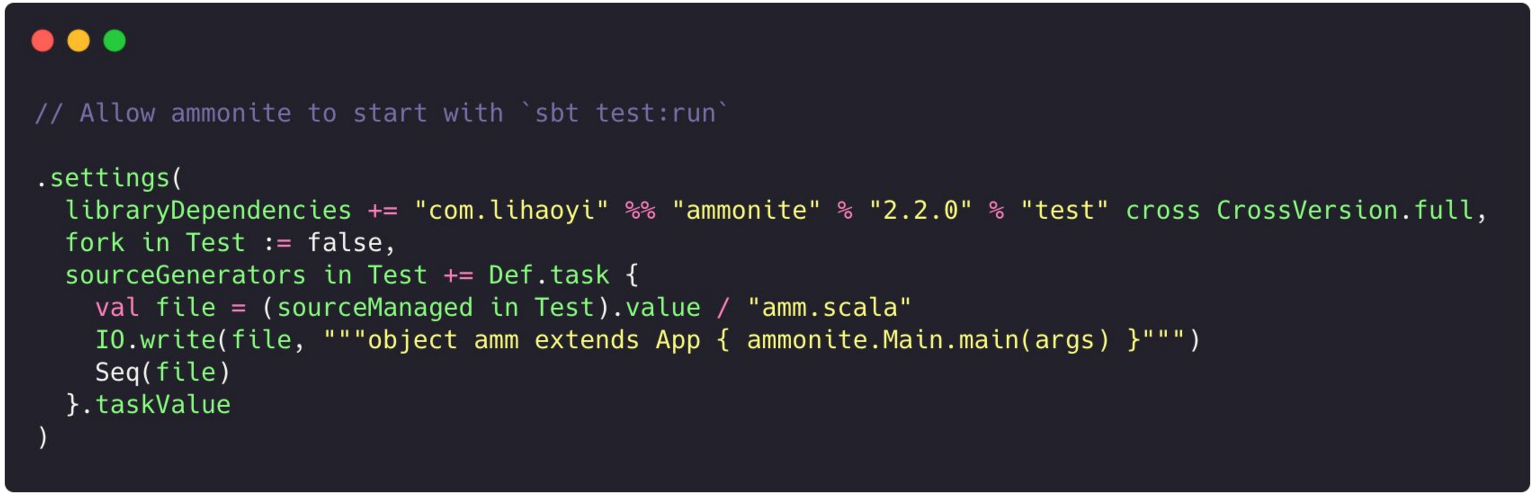
Gradle

Mill

fury

Bazel

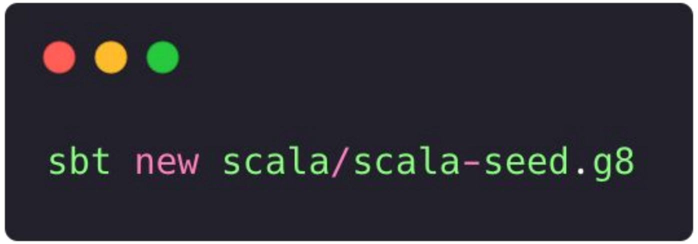
Ammonite



```
// Allow ammonite to start with `sbt test:run`

.settings(
  libraryDependencies += "com.lihaoyi" %% "ammonite" % "2.2.0" % "test" cross CrossVersion.full,
  fork in Test := false,
  sourceGenerators in Test += Def.task {
    val file = (sourceManaged in Test).value / "amm.scala"
    IO.write(file, """object amm extends App { ammonite.Main.main(args) }""")
    Seq(file)
  }.taskValue
)
```

http://ammonite.io/#Ammonite-REPL

Templates and giter8



```
sbt new scala/scala-seed.g8
```

Easy way to provide/start the bootstrap of a project

Questions ?

Exercise 1 - Bootstrap

- Create a new sbt project with latest sbt and scala versions

- Create two projects `core` and `util`, with `core` depending on `util`

- `util` module provides the pascal’s triangle solution,
  `core` module runs and use it to print the result

Exercise 2 - Use a dependency

- Create a second sbt project with latest sbt and scala versions

- This project should use the `util` module from the first project as a library to run its own version of the pascal’s triangle solution (have a look at the `publishLocal` task)

Exercise 3 - Use an external plugin

- Add the sbt-git plugin to the first project
  https://github.com/sbt/sbt-git

- From the sbt shell, make the project use git, and make a first commit adding a test case for the pascal’s triangle inside the `util` module

Exercise 4 - Running an application and debugging it

- Use https://github.com/http4s/http4s.g8 bootstrap and make sure you can make the server run

- Open a debug port on your application and make another call to check how it works (Check the options to add thanks to IntelliJ debug interface… or Google)

- Now change the url called inside the `Jokes.scala` file, restart the application and find the exception happening thanks to the debug interface

Exercice 5 - Because why not

- This is totally useless and overkill, BUT we are going to do it for fun
- Use the snapshots from these slides to read the http4s version you want to use from a `project.version.txt` file, and use it to pull the dependency