Learn you an SBT for fun and profit!

Daniel Westheide

www.danielwestheide.com

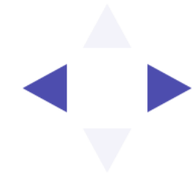
@kaffeecoder

www.innoq.com

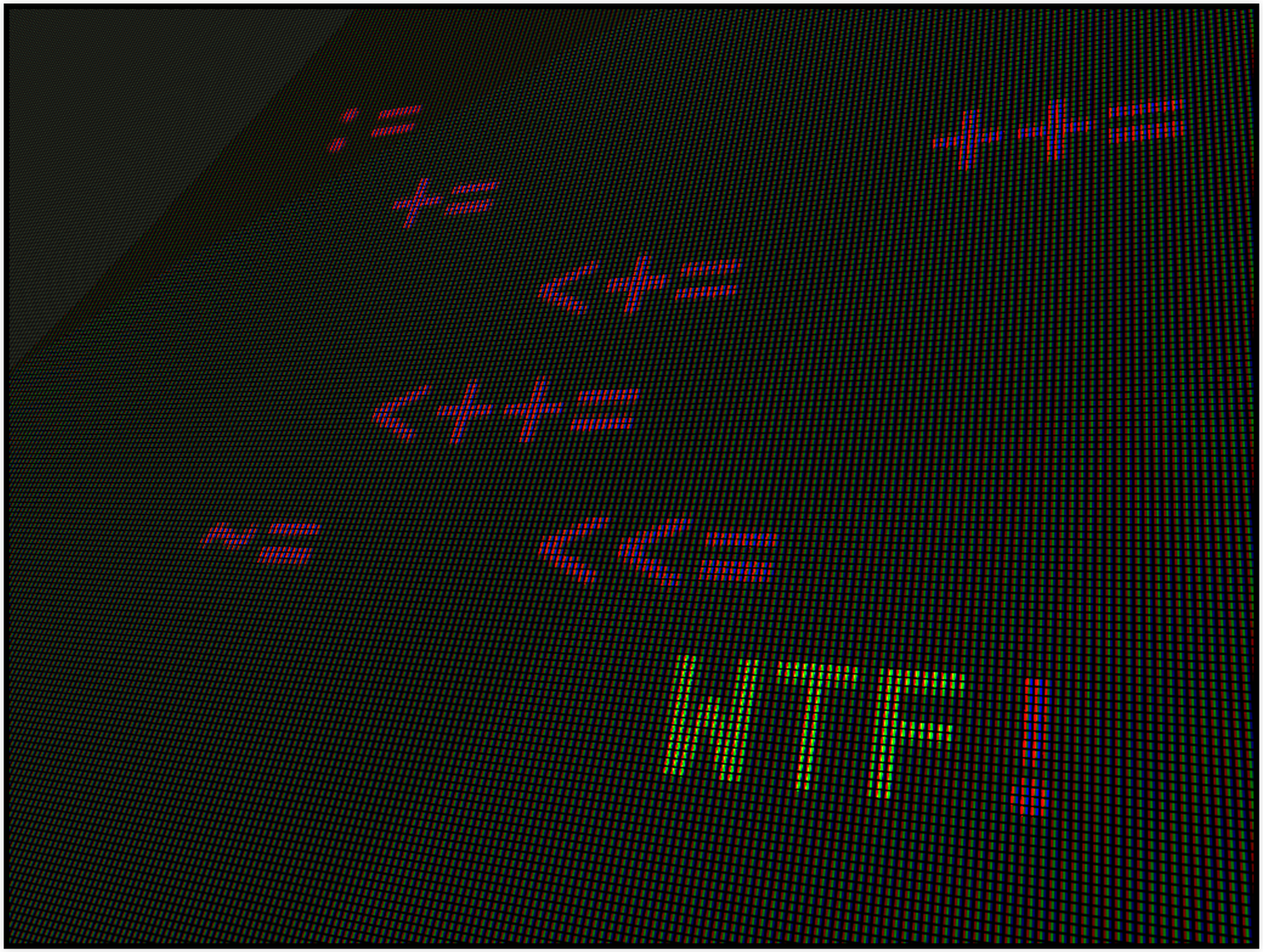
@innoq

Everybody hates build tools*

* With the exception of build engineers



SBT is not only a build tool



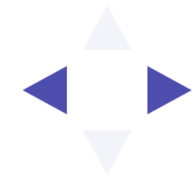
Copy/paste-driven build configuration

First, add xsbt-web-plugin to *project/plugins.sbt:*

```
addSbtPlugin("com.earldouglas" % "xsbt-web-plugin" % "0.9.0")
```

For .sbt build definitions, inject the plugin settings in *build.sbt:*

```
webSettings
```

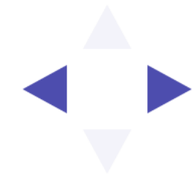


A mental model of SBT

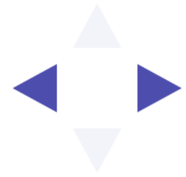
In the build.sbt, I can assign values for certain predefined attributes.

*I can compile my project by typing **sbt compile**, and test it by typing **sbt test**.*

I can add plugins by copy-pasting cryptic code from its README.



Understanding SBT

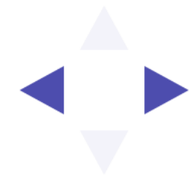


The build description

- key/value pairs
- immutable
- created by processing **build definitions**
- keys can have fixed values or computations assigned to them

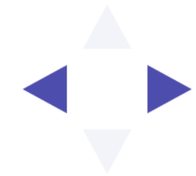
Build definitions

- consist of one or more `Settings`
- A `Setting[T]` describes a **transformation of the build description**
- Must be applied to return new build description



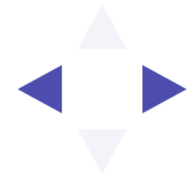
Applying a setting: before

Key	Value
name	*"scaladays-test-project"*
libraryDependencies	*List(org.scala-lang:scala-library:2.10.4)*
console	*Defaults.consoleTask*



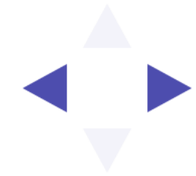
Applying a setting: after

Key	Value
name	"Case Nightmare Green"
libraryDependencies	*List(org.scala-lang:scala-library:2.10.4)*
console	*Defaults.consoleTask*



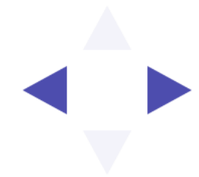
How do I create a Setting?

```
sealed abstract class SettingKey[T] {
  final def := (v: T): Setting[T] = ???
  final def +=[U](v: U)
    (implicit a: Append.Value[T, U]): Setting[T] = ???
  final def ++=[U](vs: U)
    (implicit a: Append.Values[T, U]): Setting[T] = ???
}
```



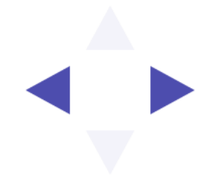
Setting dependencies

```
description :=
  s"${name.value}, kindly provided by ${organizationName.value}"
```



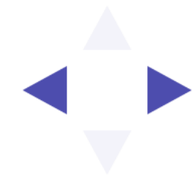
Save your SBT session

- `session list` shows a log of our SBT session
- `session save` saves it to the build.sbt file



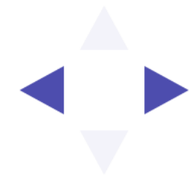
.sbt files

- One or more at the project's root level
- each provides a `Seq[Setting[_]]`
- all settings are put into one sequence
- sorted by key and dependencies



Tasks

- Computations of values or just side effects
- Executed every time their value is requested
- Triggered execution using ~
- assigned to keys just like value settings
- `TaskKey[T]` returns `Setting[Task[T]]`
- can depend on settings and other tasks



Settings are scoped

- Project axis
- Configuration axis
- Task axis

Project axis

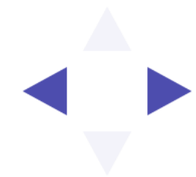
```
lazy val domain = project

lazy val infrastructure = project.dependsOn(domain).settings(
  libraryDependencies +=
    "com.typesafe" %% "akka-actor" % "2.3.3"
  )

version in ThisBuild := "0.1.0-SNAPSHOT"

organization.in(ThisBuild).:=("uk.co.laundry")
```

Use `projectName/key` to access setting scoped by project.

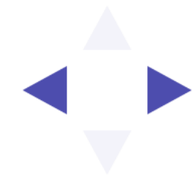


Configuration axis

```
logLevel in Compile := Level.Info

logLevel in Test := Level.Warn
```

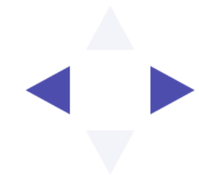
Use `configName:key` to access setting scoped by Ivy configuration



Task axis

```
initialCommands += """
import scala.concurrent.duration._
import scala.concurrent.Future
"""
initialCommands in console += """
import co.uk.laundry.domain._
"""
```

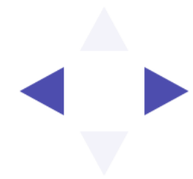
Use `taskName::key` to access setting scoped by task.



Adding new functionality

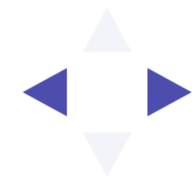
Example: Generate release notes

- Custom setting keys for release notes dir, filename and template
- Custom task keys for generating release notes and dumping to file
- Extract into Scala library code
- Leverage SBT's File, Process and IO APIs
- Input tasks with parser combinators and tab completion



Custom setting keys

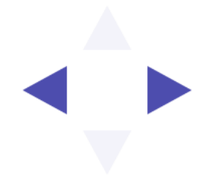
```
object ReleaseNotesKeys {
  lazy val releaseNotesDir =
    settingKey[File]("...")
  lazy val releaseNotesFileName =
    settingKey[Release => String]("...")
  lazy val releaseNotesTemplate =
    settingKey[ReleaseNotes => String]("...")
}
```



Default settings

```
object Defaults {
  def dir: Setting[File] =
      releaseNotesDir := target.value / "release-notes"
  def fileName: Setting[Release => String] =
    releaseNotesFileName := { release =>
      s"${normalizedName.value}-${release.version}.md"
  }
}
```

sbt.RichFile

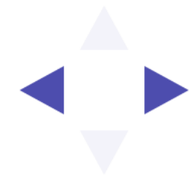


Custom task keys

```
object ReleaseNotesKeys {
  lazy val releaseNotesReleases = taskKey[List[Release]]("...")
  lazy val generateReleaseNotes = inputKey[ReleaseNotes]("...")
  lazy val dumpReleaseNotes = inputKey[File]("...")
}
```

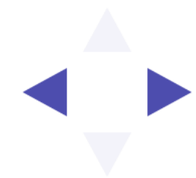
Parsing and tab completion

```
val releaseParser: Parser[(Release, Option[Release])] = {
  val releases = Implementation.releases
  val tags = releases map (_.gitTag)
  val parser = StringBasic
    .examples(tags.toSet, check = true)
  Space ~> parser map { releaseTag =>
    releases.dropWhile(_.gitTag != releaseTag) match {
      case head :: tail => (head, tail.headOption)
      case _ => sys.error("No release to generate release notes
    }
  }
}
```



Using our parser

```
generateReleaseNotes := {
  val (release, prevRelease) =
    Implementation.releaseParser.parsed
Implementation.releaseNotes(prevRelease, release)
```

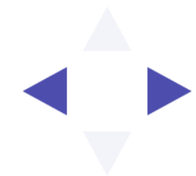


Process

```
val contribs: Stream[Contributor] =
  Process(s"git shortlog -sne $prev${release.gitTag}")
contribs.lines collect {
  case GitContributor(c) => c
}
```

Dumping the release notes with IO

```
dumpReleaseNotes := {
  val releaseNotes =
    ReleaseNotesKeys.generateReleaseNotes.evaluated
  val fileName =
    releaseNotesFileName.value(releaseNotes.release)
  val outputFile = releaseNotesDir.value / fileName
  IO.write(
    outputFile, releaseNotesTemplate.value(releaseNotes))
  outputFile
}
```



Creating a plugin

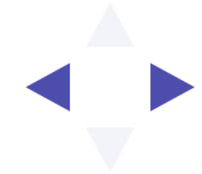
```
name := "sbt-release-notes"

organization := "com.hipstercoffee"

sbtPlugin := true
```

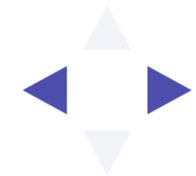
In a .sbt file in your project directory:

```
addSbtPlugin(
  "com.hipstercoffee" % "sbt-release-notes" % "0.1.0)
```



Auto plugins

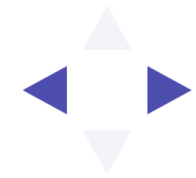
```
object ReleaseNotesPlugin extends AutoPlugin {
  override def projectSettings: Seq[Def.Setting[_]] =
    ReleaseNotes.releaseNotesSettings
}
```



Enabling auto plugins

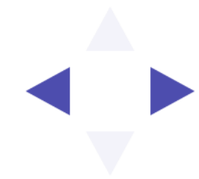
Consistent way of enabling plugins for your project:

```
lazy val root = project
  .in(file("."))
  .enablePlugins(releasenotes.ReleaseNotesPlugin)
```



Triggering automatic enabling

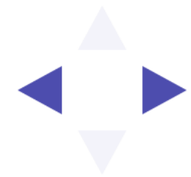
```
object ReleaseNotesPlugin extends AutoPlugin {
  override def projectSettings: Seq[Def.Setting[_]] =
    ReleaseNotes.releaseNotesSettings
  override def trigger = allRequirements
}
```



Don't reinvent the wheel

More fun and profit with existing plugins

- sbt-revolver: fast background starting and stopping of applications, and re-triggered starts
- sbt-updates: checks Maven repos for updates to your dependencies
- sbt-license-report: Reports on licenses used in your project
- sbt-git: allows using Git directly from SBT; provides Git project versioning and Git branch in prompt




Don't mess with me.
I know SBT.

Credits

- super green ninja "with lasers" by Gisela Giardano, modified by Daniel Westheide (CC BY-SA 2.0)