How to Scalafix My Code

https://github.com/hermannhueck/howto-scalafix-my-code

Abstract

This presentation with code examples gives an introduction to Scalafix. It only covers how to use Scalafix, not how to write your own scalafix rules.

It has been developed with Scala 2.12.8, sbt 1.2.8 and scalafix 0.9.5 in July 2019 and upgraded to 2.13.1, sbt 1.3.2 and scalafix 0.9.7 at 2019-09-24.

See Scalafix documentation at:
https://scalacenter.github.io/scalafix/

Agenda

1. What is Scalafix?
2. Getting Started
3. .scalafix.conf
4. Suppressing Rules in Code
5. *scalafix* Shell Command
6. Built-in Rules
7. Running (compiled) Custom Rules
8. Running Custom Rules from Source Code
9. Resources

1. What is Scalafix?

What is Scalafix? (1/3)

What is Scalafix? (1/3)

Scala code => Scala code

What is Scalafix? (1/3)

Scala code => Scala code

Scala code => Seq[LintMessage]

What is Scalafix? (2/3)

- Tool to rewrite/patch Scala code according to rewrite rules.
- Tool to lint/check Scala code according to linter rules.

What is Scalafix? (3/3) - more details

- Based on *Scalameta* and *SemanticDB*.
- Scalafix integrates nicely into sbt (sbt plugin).
- Scalafix also peovides a CLI.
- The rules to apply can be specified interactively at the (sbt or shell) prompt or in a configuration file.
- Scalafix rules are either syntactic or semantic.
- Syntactic rules work out of the box (when the scalafix plugin is installed). They don't require compilation.
- Semantic rules need information from SemanticDB. They require compilation with the *scalafixSemanticdb* compiler plugin avtivated (and an additional scalac option).

2. Getting Started

Scalafix User Guide:

https://scalacenter.github.io/scalafix/docs/users/installation.html

Scalafix - sbt plugin

project/plugins.sbt

```
addSbtPlugin("ch.epfl.scala" % "sbt-scalafix" % "0.9.5")
```

- With this plugin syntactic rules work out of the box.

Syntactic rule *ProcedureSyntax*

```
// src/main/scala/tofix/HelloApp.scala
package tofix

import scala.util.Try    // <<---

object HelloApp extends App {

  def hello() {    // <<---
    println("hello")
  }

  hello()
}
```

Syntactic rule *ProcedureSyntax*

```
// src/main/scala/tofix/HelloApp.scala
package tofix

import scala.util.Try    // <<---

object HelloApp extends App {

  def hello() {    // <<---
    println("hello")
  }

  hello()
}
```

This code has two problems:

- Ununsed import
- Discouraged procedure syntax

Check code against rule *ProcedureSyntax*

... without changing it:

```
sbt:howto-scalafix-my-code> scalafix --check --stdout ProcedureSyntax
--- .../howto-scalafix-my-code/src/main/scala/tofix/HelloApp.scala
+++ <expected fix>
@@ -4,7 +4,7 @@

 object HelloApp extends App {

-    def hello() {    // <<---
+    def hello(): Unit = {    // <<---
         println("hello")
     }

[error] (Compile / scalafix) scalafix.sbt.ScalafixFailed: TestError
```

- *--check*: lets scalafix fail, if it finds procedure syntax in a source file
- *--stdout*: prints a patch to stdout without applying it to the file

Fix code against rule *ProcedureSyntax*

... modifying the source file(s):

```
sbt:howto-scalafix-my-code> scalafix ProcedureSyntax
[success] Total time: 0 s, completed 04.08.2019, 16:45:25
```

Fix code against rule *ProcedureSyntax*

... modifying the source file(s):

```
sbt:howto-scalafix-my-code> scalafix ProcedureSyntax
[success] Total time: 0 s, completed 04.08.2019, 16:45:25
```

After the application of the patch procedure syntax has been replaced by regular method syntax:

```
// src/main/scala/tofix/HelloApp.scala
package tofix

import scala.util.Try    // <<---

object HelloApp extends App {

  def hello(): Unit = {    // <<---
    println("hello")
  }

  hello()
}
```

Enable semantic rules

build.sbt

```
addCompilerPlugin(scalafixSemanticdb)
scalacOptions += "-Yrangepos"   // use range positions for syntax trees
```

The compiler generates a *.semanticdb* file for each source file.

```
howto-scalafix-my-code $ find . -name '*.semanticdb'
./target/scala-2.12/classes/META-INF/semanticdb/src/main/..
                                        ..scala/tofix/HelloApp.scala.semanticdb
...
```

Enable semantic rules

build.sbt

```
addCompilerPlugin(scalafixSemanticdb)
scalacOptions += "-Yrangepos"   // use range positions for syntax trees
```

The compiler generates a *.semanticdb* file for each source file.

```
howto-scalafix-my-code $ find . -name '*.semanticdb'
./target/scala-2.12/classes/META-INF/semanticdb/src/main/..
                                        ..scala/tofix/HelloApp.scala.semanticdb
...
```

.semanticdb files contain information about the AST (abstract syntax tree) in binary protobuf format.

Semantic rule *RemoveUnused*

- Some rules need rule-specific scalac options.
- *RemoveUnused* needs *-Ywarn-unused*.

build.sbt

```
scalacOptions += "-Ywarn-unused"
```

Check code against rule *RemoveUnused*

... without changing it:

```
sbt:howto-scalafix-my-code> scalafix --check --stdout RemoveUnused
--- .../howto-scalafix-my-code/src/main/scala/tofix/HelloApp.scala
+++ <expected fix>
@@ -1,6 +1,5 @@
 package tofix

-import scala.util.Try    // <<---

 object HelloApp extends App {

[error] (Compile / scalafix) scalafix.sbt.ScalafixFailed: TestError
[error] Total time: 0 s, completed 04.08.2019, 17:53:06
```

Fix code against rule *RemoveUnused*

... modifying the source file(s):

```
sbt:howto-scalafix-my-code> scalafix RemoveUnused
[success] Total time: 1 s, completed 04.08.2019, 18:02:20
```

Fix code against rule *RemoveUnused*

... modifying the source file(s):

```
sbt:howto-scalafix-my-code> scalafix RemoveUnused
[success] Total time: 1 s, completed 04.08.2019, 18:02:20
```

After the application of the patch the unused import is gone:

```
// src/main/scala/tofix/HelloApp.scala
package tofix

    // <<---

object HelloApp extends App {

  def hello(): Unit = {    // <<---
    println("hello")
  }

  hello()
}
```

3. .scalafix.conf

https://scalacenter.github.io/scalafix/docs/users/configuration.html

.scalafix.conf

- In the previous examples we specified the rules at the sbt prompt.
- Instead we can use a config file in HOCON syntax (https://github.com/lightbend/config).
- Default file: *.scalafix.conf* in the project root directory

.scalafix.conf

- In the previous examples we specified the rules at the sbt prompt.
- Instead we can use a config file in HOCON syntax (https://github.com/lightbend/config).
- Default file: *.scalafix.conf* in the project root directory

```
// .scalafix.conf

rules = [
  ProcedureSyntax
  RemoveUnused
]
```

.scalafix.conf

- In the previous examples we specified the rules at the sbt prompt.
- Instead we can use a config file in HOCON syntax (https://github.com/lightbend/config).
- Default file: *.scalafix.conf* in the project root directory

```
// .scalafix.conf

rules = [
  ProcedureSyntax
  RemoveUnused
]
```

Rules no longer specified at the sbt prompt:

```
sbt:howto-scalafix-my-code> scalafix --check --stdout
--- .../howto-scalafix-my-code/src/main/scala/tofix/HelloApp.scala
+++ <expected fix>
...
```

Disable checking unused imports

```
// .scalafix.conf

rules = [
  ProcedureSyntax
  RemoveUnused
]

RemoveUnused.imports = false // ignore unused imports
```

Scalafix will check for unused local definitions and unused private members, but will skip unused imports.

Disable checking unused imports

```
// .scalafix.conf

rules = [
  ProcedureSyntax
  RemoveUnused
]

RemoveUnused.imports = false // ignore unused imports
```

Scalafix will check for unused local definitions and unused private members, but will skip unused imports.

For more information see:

https://scalacenter.github.io/scalafix/docs/rules/RemoveUnused.html

Rule *DisableSyntax*

- The *RemoveUnused* rule was opt-out by default. If enabled all sub rules (*RemoveUnused.import*, *RemoveUnused.privates*. *RemoveUnused.locals*) are also enabled.
  - Sub rules must be disabled explicitly.
- The *DisableSyntax* rule is opt-in by default. If enabled all sub rules are still disabled.
  - Sub rules must be enabled explicitly.

Rule *DisableSyntax*

- The *RemoveUnused* rule was opt-out by default. If enabled all sub rules (*RemoveUnused.import*, *RemoveUnused.privates*. *RemoveUnused.locals*) are also enabled.
  - Sub rules must be disabled explicitly.
- The *DisableSyntax* rule is opt-in by default. If enabled all sub rules are still disabled.
  - Sub rules must be enabled explicitly.

```
// .scalafix.conf

rules = [
  ProcedureSyntax
  RemoveUnused
  DisableSyntax
]

RemoveUnused.imports = false // ignore unused imports
DisableSyntax.noThrows = true // disallow throwing exceptions
```

Check rule *DisableSyntax*

```
sbt:howto-scalafix-my-code> scalafix --check --stdout
[info] Running scalafix ...
...
[error] .../tofix/Throwing.scala:14:7: error: [DisableSyntax.throw] exceptions sho
[error]       throw new IllegalArgumentException("string is empty")
[error]       ^^^^^
[error] .../tofix/Throwing.scala:18:22: error: [DisableSyntax.throw] exceptions sh
[error]         case None => throw new NumberFormatException("not an Int")
[error]                      ^^^^^
[error] .../tofix/Throwing.scala:23:9: error: [DisableSyntax.throw] exceptions sho
[error]         throw new RuntimeException("negative Int")
[error]         ^^^^^
[error] .../tofix/Throwing.scala:31:5: error: [DisableSyntax.throw] exceptions sho
[error]     throw new IllegalArgumentException("no args specified")
[error]     ^^^^^
[error] (Compile / scalafix) scalafix.sbt.ScalafixFailed: TestError LinterError
```

Four code locations in *Throwing.scala* have *throw* statements. Scalafix conplains about all of them.

Scalafix does not provide patches in this case. These problems cannot be fixed automatically by the tool. They need manual redesign of the code.

4. Suppressing Rules in Code

https://scalacenter.github.io/scalafix/docs/users/suppression.html

Suppress rules using annotations

Suppress scalafix check with annotation *@SuppressWarnings*:

```
// src/main/scala/tofix/Throwing.scala
...
@SuppressWarnings(Array("scalafix:DisableSyntax.throw"))
def run(args: List[String]) =
  if (args.length == 0) {
    throw new IllegalArgumentException("no args specified")
  } else {
    checkNumber(args.head)
  }
```

- This suppresses the rule *DisableSyntax.noThrows* for the annotated method, class or object.
- *@SuppressWarnings(Array("all"))* suppresses all rules for the annotated method, class or object.

Suppress rule using *scalafix:ok*

Suppress scalafix check for a code location with comment *scalafix:ok*:

```
// src/main/scala/tofix/Throwing.scala
...
  if (string.isEmpty()) {
    throw new IllegalArgumentException("string is empty") // scalafix:ok
  } else {
    ...
```

Suppress rule using *scalafix:ok*

Suppress scalafix check for a code location with comment *scalafix:ok*:

```
// src/main/scala/tofix/Throwing.scala
...
  if (string.isEmpty()) {
    throw new IllegalArgumentException("string is empty") // scalafix:ok
  } else {
    ...
```

- The *scalafix:ok* comment can be placed after the expression to disable.
- The comment can also be placed before the expression to disable.

Suppress code regions using *scalafix:off* and *scalafix:on*

Suppress scalafix for a code region:

```
// src/main/scala/tofix/Throwing.scala
...
  // scalafix:off
  val num = string.toMaybeInt match {
    case None => throw new NumberFormatException("not an Int")
    case Some(int) => int
  }

  if (num < 0) {
    throw new RuntimeException("negative Int")
  }
  // scalafix:on
```

5. *scalafix* CLI

https://scalacenter.github.io/scalafix/docs/users/installation.html#command-line

scalafix CLI

- can be installed with *coursier*
- Install *coursier* on your system. See: https://get-coursier.io/docs/cli-overview
- Install *scalafix* (e.g. to your *~/bin* directory):

```
# adjust scala version and scalafix version as needed

coursier bootstrap ch.epfl.scala:scalafix-cli_2.12.10:0.9.7 \
                                -f --main scalafix.cli.Cli -o ~/bin/scalafix

./scalafix --version # Should say 0.9.7
./scalafix --help # prints a long list of flags
```

- Hint: Use flag *--scalac-options* to set the *scalacOptions* you normally set in your *build.sbt*.

Using CLI ...

... to check *HelloApp.scala*

```
howto-scalafix-my-code $ \
  scalafix \
    --check # only check, don't fix \
    --stdout # write patch to stdout \
    --scalac-options -Yrangepos # scalac-option: range positions for syntax trees
    --scalac-options -Ywarn-unused # scalac-option: enable 'unused' warnings \
    --auto-classpath # automatically scan for directories with META-INF/semanticdb
    --files src/main/scala/tofix/HelloApp.scala # files to fix  \
    --rules RemoveUnused ProcedureSyntax # rules to apply

--- .../howto-scalafix-my-code/src/main/scala/tofix/HelloApp.scala
+++ <expected fix>
@@ -1,7 +1,6 @@
 package tofix
+    // <<---

-import scala.util.Try    // <<---
-
 object HelloApp extends App {

     def hello() {    // <<---
```

6. Built-in Rules

https://scalacenter.github.io/scalafix/docs/rules/overview.html

Built-in rules - syntactic and semantic rules

Scalafix comes with a small set of built-in rules.
Rules are either syntactic or semantic.

- **Syntactic**: the rule can run directly on source code without compilation. Syntactic rules are simple to run but they can only do limited code analysis since they do not have access to information such as symbols and types.
- **Semantic**: the rule requires input sources to be compiled beforehand with the Scala compiler and the SemanticDB compiler plugin enabled. Semantic rules are more complicated to run but they can do more advanced code analysis based on symbols and types.

Built-in semantic rules - overview

Name	Description
RemoveUnused	Removes unused imports and terms that reported by the compiler under -Ywarn-unused
NoAutoTupling	Inserts explicit tuples for adapted argument lists for compatibility with -Yno-adapted-args

Built-in syntactic rules - overview

Name	Description
ProcedureSyntax	Replaces deprecated procedure syntax with explicit ': Unit ='
NoValInForComprehension	Removes deprecated val inside for-comprehension binders
LeakingImplicitClassVal	Adds 'private' to val parameters of implicit value classes
DisableSyntax	Reports an error for disabled features such as var or XML literals

Semantic rule *RemoveUnused*

- Enable the Scala compiler option *-Ywarn-unused*. sbt: *scalacOptions += "-Ywarn-unused"*
- Disable *-Xfatal-warnings*. This is required so the compiler warnings do not fail the build before running Scalafix.

Configuration and Defaults:

Name	Type	Default	Description
RemoveUnused.imports	Boolean	true	Remove unused imports
RemoveUnused.privates	Boolean	true	Remove unused private members
RemoveUnused.locals	Boolean	true	Remove unused local definitions

https://scalacenter.github.io/scalafix/docs/rules/RemoveUnused.html

Semantic rule *NoAutoTupling*

- Enable the Scala compiler option *-Ywarn-adapted-args*. sbt: *scalacOptions += "-Ywarn-adapted-args"*
- Disable *-Xfatal-warnings*. This is required so the compiler warnings do not fail the build before running Scalafix.

Configuration and Defaults:

NoAutoTupling has no sub-rules.

https://scalacenter.github.io/scalafix/docs/rules/NoAutoTupling.html

Syntactic rule *ProcedureSyntax*

"Procedure syntax" is a deprecated Scala feature that allows methods to leave out the result type and assignment character =. This rule replaces procedure syntax with an explicit : Unit result type and adds the = character.

We've seen this in the "Getting Started" section.

Configuration and Defaults:

ProcedureSyntax has no sub-rules.

https://scalacenter.github.io/scalafix/docs/rules/ProcedureSyntax.html

Syntactic rule *NoValInForComprehension*

Removes *val* keyword from definitions in for-comprehension, e.g. here:

```
for {
  n <- List(1, 2, 3)
  val inc = n + 1 // <<--- 'val' deprecated in for comprehension
} yield inc
```

Configuration and Defaults:

NoValInForComprehension has no sub-rules.

https://scalacenter.github.io/scalafix/docs/rules/NoValInForComprehension.html

Syntactic rule *LeakingImplicitClassVal*

Non-private val fields of implicit classes leak as publicly accessible extension methods. This rule adds the private access modifier on the field of implicit value classes in order to prevent direct access.

```
implicit class XtensionVal(val str: String) extends AnyVal { // <<--- str should
  def doubled: String = str + str
}

"message".str // compiles, but shouldn't
```

Configuration and Defaults:

LeakingImplicitClassVal has no sub-rules.

https://scalacenter.github.io/scalafix/docs/rules/LeakingImplicitClassVal.html

Syntactic rule *DisableSyntax*

This rules allows you to disable certain syntax features which are legal Scala but discouraged such as *asInstanceOf*, *null*s, *throw*ing exceptions etc.

Merely enabling this rule disables nothing. All sub-rules are false by default. Additionally you have to enable the respective sub-rule you are interested in.

We've seen this with the sub-rule *DisableSyntax.noThrows* in the "Getting Started" section.

Configuration and Defaults: see next slide

https://scalacenter.github.io/scalafix/docs/rules/DisableSyntax.html

Sub-Rule	Type	Default	Disallowed Syntax
noVars	Boolean	false	Usage of vars
noThrows	Boolean	false	Usage of throws
noNulls	Boolean	false	Usage of null
noReturns	Boolean	false	Usage of return
noWhileLoops	Boolean	false	Usage of while loops
noAsInstanceOf	Boolean	false	Usage of asInstanceOf[T]
noIsInstanceOf	Boolean	false	Usage of isInstanceOf[T]
noXml	Boolean	false	Usage of XML literals
noDefaultArgs	Boolean	false	Usage of method params with default args
noFinalVal	Boolean	false	Usage of 'final' for 'val' definitions
noFinalize	Boolean	false	Overriding finalize
noValPatterns	Boolean	false	pat. mat. in val assignment with non-tuple pat
noUniversalEquality	Boolean	false	Usage of `==` (universal equality)
noUniversalEqualityMessage	String	see below: 1	Reporter message for noUniversalEquality
regex	see below: 2	[]	When regex in source file matches

1.) "== and != are unsafe since they allow comparing two unrelated types"

2.) List[CustomMessage[Either[Regex, Pattern]]]

7. Running (compiled) Custom Rules

https://scalacenter.github.io/scalafix/docs/users/installation.html#run-custom-rules

Running (compiled) Custom Rules

- *scalafix* comes with built-in rules (*RemoveUnused*, *ProcedureSyntax* etc.) which you can run out of the box once *scalafix* is installed and set up correctly.
- Additionally you can apply compiled rules published to Maven Central.
- You can specify such a rules with *scalafixDependencies* analogously to *libraryDependencies*.

build.sbt

```
scalafixDependencies ++= Seq(
  // for rules 'MissingFinal' and 'Disable'
  // source at: https://github.com/vovapolu/scaluzzi
  "com.github.vovapolu" %% "scaluzzi" % "0.1.2",
),
```

Check rule *MissingFinal*

If you have an ADT (sealed trait and derived case classes) in one source file, the case classes should carry a *final* modifier.

Check rule *MissingFinal*

If you have an ADT (sealed trait and derived case classes) in one source file, the case classes should carry a *final* modifier.

```
sealed trait MyADT extends Product with Serializable

case class Foo(s: String) extends MyADT    // <<--- missing final
case class Bar(i: Int) extends MyADT
case class Baz(b: Boolean) extends MyADT
```

Check rule *MissingFinal*

If you have an ADT (sealed trait and derived case classes) in one source file, the case classes should carry a *final* modifier.

```
  sealed trait MyADT extends Product with Serializable

  case class Foo(s: String) extends MyADT    // <<--- missing final
  case class Bar(i: Int) extends MyADT
  case class Baz(b: Boolean) extends MyADT
```

```
sbt:howto-scalafix-my-code> scalafix --check --stdout MissingFinal
[info] Running scalafix ...
--- .../howto-scalafix-my-code/src/main/scala/scaluzzi/ExampleMissingFinal.scala
+++ <expected fix>
@@ -4,7 +4,7 @@

     sealed trait MyADT extends Product with Serializable

-    case class Foo(s: String) extends MyADT
-    case class Bar(i: Int) extends MyADT
-    case class Baz(b: Boolean) extends MyADT
+    final case class Foo(s: String) extends MyADT
+    final case class Bar(i: Int) extends MyADT
+    final case class Baz(b: Boolean) extends MyADT
 }
[error] (Compile / scalafix) scalafix.sbt.ScalafixFailed: TestError
```

8. Running Custom Rules from Source Code

https://scalacenter.github.io/scalafix/docs/developers/tutorial.html#run-the-rule-from-source-code

Running Custom Rules from Source Code

scalafix --rules=[source-url]

where source url can be:

- a file url
- a http/https url
- a github repo

Source Rule by file URL

```
> scalafix --check --stdout --rules=file:./rule-files/MissingFinal.scala
```

Source Rule by file URL

```
> scalafix --check --stdout --rules=file:./rule-files/MissingFinal.scala
```

```
sbt:howto-scalafix-my-code> scalafix --check --stdout --rules=file:./rule-files/Mi
[info] Running scalafix ...
...
--- .../howto-scalafix-my-code/src/main/scala/scaluzzi/ExampleMissingFinal.scala
+++ <expected fix>
@@ -4,7 +4,7 @@

     sealed trait MyADT extends Product with Serializable

-    case class Foo(s: String) extends MyADT    // <<--- missing final
-    case class Bar(i: Int) extends MyADT
-    case class Baz(b: Boolean) extends MyADT
+    final case class Foo(s: String) extends MyADT    // <<--- missing final
+    final case class Bar(i: Int) extends MyADT
+    final case class Baz(b: Boolean) extends MyADT
 }
[error] (Compile / scalafix) scalafix.sbt.ScalafixFailed: TestError
```

Source Rule by HTTP URL

```
> scalafix --check --stdout --rules=...
  ...https://raw.githubusercontent.com/vovapolu/scaluzzi/master...
  .../scalafix/rules/src/main/scala/scalafix/internal/scaluzzi/MissingFinal.scala
```

Source Rule by HTTP URL

```
> scalafix --check --stdout --rules=...
  ...https://raw.githubusercontent.com/vovapolu/scaluzzi/master...
  .../scalafix/rules/src/main/scala/scalafix/internal/scaluzzi/MissingFinal.scala
```

```
sbt:howto-scalafix-my-code> scalafix --check --stdout --rules=https://raw.githubus
[info] Running scalafix ...
...
--- .../howto-scalafix-my-code/src/main/scala/scaluzzi/ExampleMissingFinal.scala
+++ <expected fix>
@@ -4,7 +4,7 @@

     sealed trait MyADT extends Product with Serializable

-    case class Foo(s: String) extends MyADT    // <<--- missing final
-    case class Bar(i: Int) extends MyADT
-    case class Baz(b: Boolean) extends MyADT
+    final case class Foo(s: String) extends MyADT    // <<--- missing final
+    final case class Bar(i: Int) extends MyADT
+    final case class Baz(b: Boolean) extends MyADT
 }
[error] (Compile / scalafix) scalafix.sbt.ScalafixFailed: TestError
```

Source Rule by github URL

URL pattern:

```
> scalafix --check --stdout --rules=github:[user]/[repo]/[rule-name]
```

This expects the rule source file to be located in the package *fix* under *scalafix/rules/src/main/scala*

Source Rule by github URL

URL pattern:

```
> scalafix --check --stdout --rules=github:[user]/[repo]/[rule-name]
```

This expects the rule source file to be located in the package *fix* under *scalafix/rules/src/main/scala*

```
> scalafix --check --stdout --rules=github:vovapolu/scaluzzi/MissingFinal

[error] (Compile / scalafix) scalafix.sbt.InvalidArgument: 404 - not found ...
   ...https://raw.githubusercontent.com/vovapolu/scaluzzi...
   .../master/scalafix/rules/src/main/scala/fix/MissingFinal.scala
```

The rule *MissingFinal* is not located at the standard location and therefore not found using the github URL.

Upgrade your *cats* code ...

... from *cats-0.9.0* to *cats-1.0.0*:

```
sbt scalafix github:typelevel/cats/v1.0.0
```

Scalafix rules for cats:

https://github.com/typelevel/cats/tree/master/scalafix

Pros and Cons

Pros

- avoids the effort to publish the rule to Maven Central or Sonatype
- more flexible during development and test

Cons

- Inflexible. Rules must be implemented in a single source file.
- No dependencies. Rules can only use the Scalafix public API.
- Slow. Rule is re-compiled on every invocation so it's not great for interactive usage.
- No tab completion in the sbt shell. Users need to manually type the path to the source file.

How to write your own rules

Scalafix developers guide:

https://scalacenter.github.io/scalafix/docs/developers/setup.html

9. Resources

Resources

- Code and Slides of this Talk:
  https://github.com/hermannhueck/howto-scalafix-my-code
- Scalafix Home/Docs:
  https://scalacenter.github.io/scalafix/
- Scalafix on Github:
  https://github.com/scalacenter/scalafix
- Refactoring with scalafix and scala.meta
  Talk by Ólafur Páll Geirsson at ScalaSphere 2017
  https://www.youtube.com/watch?v=7I18pJ6orrI
- Scalafix Workshop by Gabriele Petronella at Scala World 2017
  https://www.youtube.com/watch?v=uaMWvkCJM_E
- DevInsideYou: Getting started with #Scalafix
  https://www.youtube.com/watch?v=Xl8oOmFNGgQ

Thank You

Q & A

https://github.com/hermannhueck/howto-scalafix-my-code