Metaprogramming 2.0

Eugene Burmako (@xeno_by)

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

11 May 2016

scala.meta is a dream

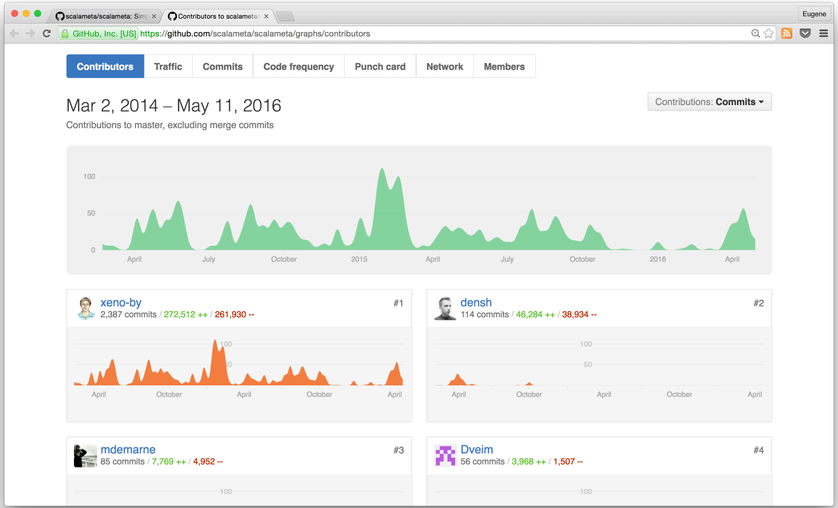
Easy Metaprogramming For Everyone!

Eugene Burmako (@xeno_by)
Denys Shabalin (@den_sh)

École Polytechnique Fédérale de Lausanne
http://scalameta.org/

17 June 2014

scala.meta is an active project



Mar 2, 2014 – May 11, 2016

Contributions to master, excluding merge commits

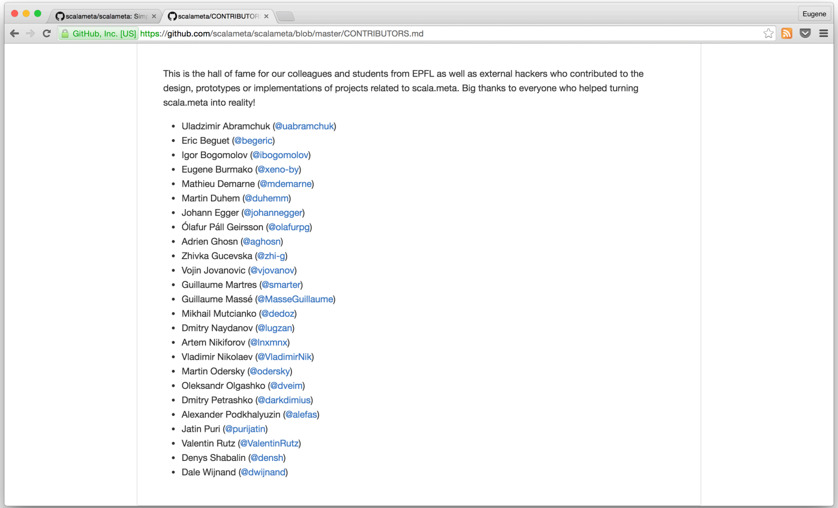
xeno-by #1
2,387 commits / 272,512 ++ / 261,930 --

densh #2
114 commits / 46,284 ++ / 38,934 --

mdemarne #3
85 commits / 7,769 ++ / 4,952 --

Dveim #4
56 commits / 3,968 ++ / 1,507 --

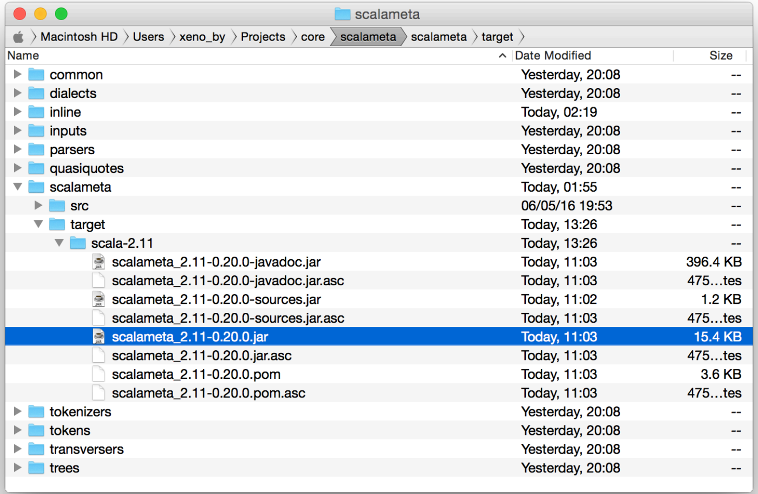
scala.meta is a community


scalameta/scalameta: Simp
scalameta/CONTRIBUTOR
Eugene
GitHub, Inc. [US] https://github.com/scalameta/scalameta/blob/master/CONTRIBUTORS.md

This is the hall of fame for our colleagues and students from EPFL as well as external hackers who contributed to the design, prototypes or implementations of projects related to scala.meta. Big thanks to everyone who helped turning scala.meta into reality!

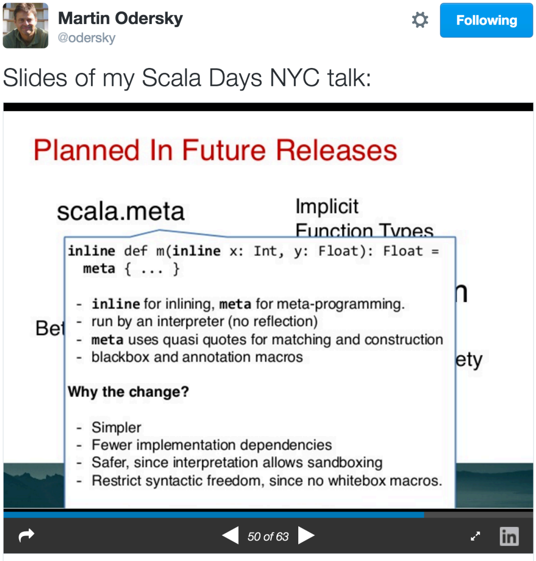
- Uladzimir Abramchuk (@uabramchuk)
- Eric Beguet (@begeric)
- Igor Bogomolov (@ibogomolov)
- Eugene Burmako (@xeno-by)
- Mathieu Demarne (@mdemarne)
- Martin Duhem (@duhemm)
- Johann Egger (@johannegger)
- Ólafur Páll Geirsson (@olafurpg)
- Adrien Ghosn (@aghosn)
- Zhivka Gucevska (@zhi-g)
- Vojin Jovanovic (@vjovanov)
- Guillaume Martres (@smarter)
- Guillaume Massé (@MasseGuillaume)
- Mikhail Mutcianko (@dedoz)
- Dmitry Naydanov (@lugzan)
- Artem Nikiforov (@lnxmnx)
- Vladimir Nikolaev (@VladimirNik)
- Martin Odersky (@odersky)
- Oleksandr Olgashko (@dveim)
- Dmitry Petrashko (@darkdimius)
- Alexander Podkhalyuzin (@alefas)
- Jatin Puri (@purijatin)
- Valentin Rutz (@ValentinRutz)
- Denys Shabalin (@densh)
- Dale Wijnand (@dwijnand)

scala.meta is a product


scalameta
Macintosh HD › Users › xeno_by › Projects › core › scalameta › scalameta › target ›

Name	Date Modified	Size
▶ common	Yesterday, 20:08	--
▶ dialects	Yesterday, 20:08	--
▶ inline	Today, 02:19	--
▶ inputs	Yesterday, 20:08	--
▶ parsers	Yesterday, 20:08	--
▶ quasiquotes	Yesterday, 20:08	--
▼ scalameta	Today, 01:55	--
▶ src	06/05/16 19:53	--
▼ target	Today, 13:26	--
▼ scala-2.11	Today, 13:26	--
scalameta_2.11-0.20.0-javadoc.jar	Today, 11:03	396.4 KB
scalameta_2.11-0.20.0-javadoc.jar.asc	Today, 11:03	475…tes
scalameta_2.11-0.20.0-sources.jar	Today, 11:02	1.2 KB
scalameta_2.11-0.20.0-sources.jar.asc	Today, 11:03	475…tes
scalameta_2.11-0.20.0.jar	Today, 11:03	15.4 KB
scalameta_2.11-0.20.0.jar.asc	Today, 11:03	475…tes
scalameta_2.11-0.20.0.pom	Today, 11:03	3.6 KB
scalameta_2.11-0.20.0.pom.asc	Today, 11:03	475…tes
▶ tokenizers	Yesterday, 20:08	--
▶ tokens	Yesterday, 20:08	--
▶ transversers	Yesterday, 20:08	--
▶ trees	Yesterday, 20:08	--

scala.meta is officially endorsed



Martin Odersky
@odersky

Following

Slides of my Scala Days NYC talk:

Planned In Future Releases

scala.meta

Implicit
Function Types

Bet

n

ety

```
inline def m(inline x: Int, y: Float): Float =
  meta { ... }
```

- **inline** for inlining, **meta** for meta-programming.
- run by an interpreter (no reflection)
- **meta** uses quasi quotes for matching and construction
- blackbox and annotation macros

Why the change?

- Simpler
- Fewer implementation dependencies
- Safer, since interpretation allows sandboxing
- Restrict syntactic freedom, since no whitebox macros.

50 of 63

Part 1: What can scala.meta do?

ScalaDays 2015

- Experimentation's temporarily on hold, were now pushing for 0.1
- Main focus of 0.1 is making scala.meta trees publicly available

ScalaDays 2016

- 3k commits and 19 milestones later, we're almost there
- Today we have published our first beta release: v0.20.0
- We hope v1.0.0 to follow in the near future

Supported functionality

Feature-complete for v1.0.0:

- Parsing
- Quasiquotes
- Tokenization
- Prettyprinting

Future releases

Planned for v2.0.0:

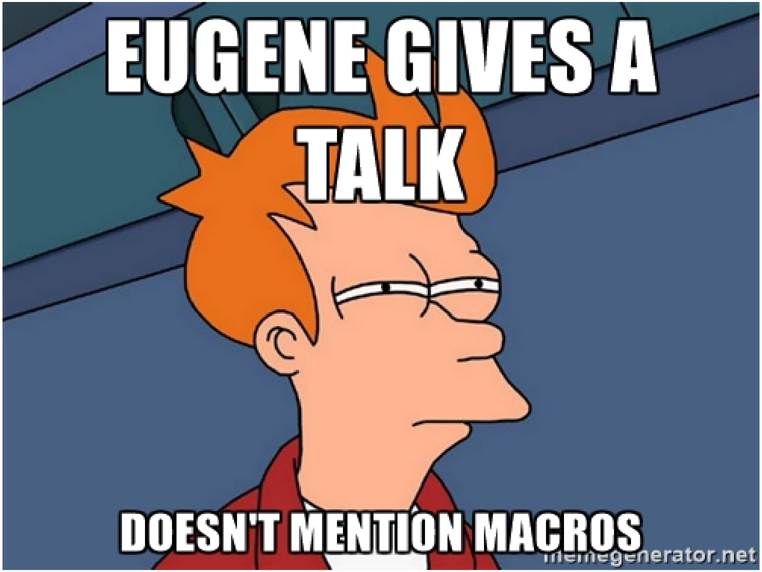
- AST persistence
- Name resolution
- Typechecking
- Implicit inference

Part 2: Live demo

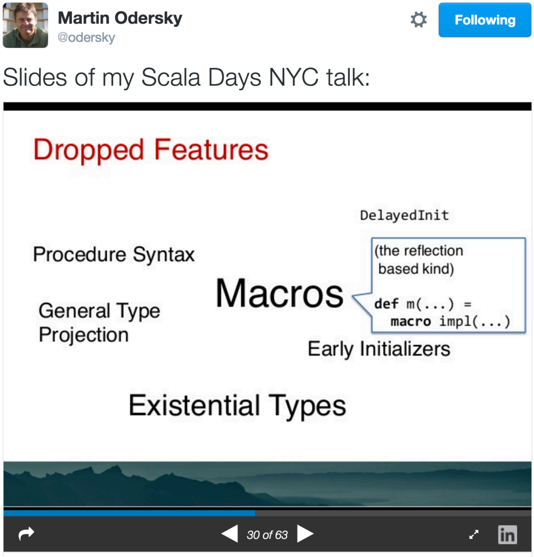
Summary

- Parsing different dialects of Scala
- Type-safe quasiquotes
- Tokenization for advanced tooling
- Prettyprinting that respects formatting and comments

Part 3: But what about macros?


EUGENE GIVES A TALK
DOESN'T MENTION MACROS
memegenerator.net

Macros are dead



Martin Odersky
@odersky

Following

Slides of my Scala Days NYC talk:

Dropped Features

DelayedInit

Procedure Syntax

Macros

(the reflection based kind)

```
def m(...) =
  macro impl(...)
```

General Type Projection

Early Initializers

Existential Types

30 of 63

Macros are bad

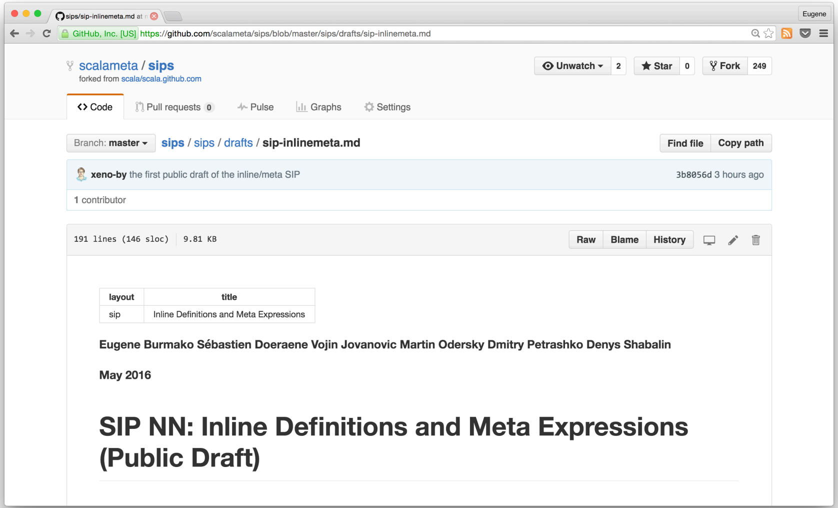
- Hard to write
- Don't work with tools well
- Hopelessly entangled with scalac

Macros are great

- Enable unique use cases
- Frequently used by library authors
- Most of the complexity is incidental

Long live macros

The new future for macros


sips/sip-inlinemeta.md at
Eugene
GitHub, Inc. [US] https://github.com/scalameta/sips/blob/master/sips/drafts/sip-inlinemeta.md
scalameta / sips
forked from scala/scala.github.com
Unwatch 2
Star 0
Fork 249
Code
Pull requests 0
Pulse
Graphs
Settings
Branch: master
sips / sips / drafts / sip-inlinemeta.md
Find file
Copy path
xeno-by the first public draft of the inline/meta SIP
3b8056d 3 hours ago
1 contributor
191 lines (146 sloc) | 9.81 KB
Raw
Blame
History

layout	title
sip	Inline Definitions and Meta Expressions

Eugene Burmako Sébastien Doeraene Vojin Jovanovic Martin Odersky Dmitry Petrashko Denys Shabalin

May 2016

SIP NN: Inline Definitions and Meta Expressions (Public Draft)

The new future for macros

- SIP NN: Inline Definitions and Meta Expressions
- We got to the essence of macros and found two orthogonal concepts
- `inline` for inlining and `meta` for metaprogramming

Old macros

```
class Table[T](val query: Query[T]) {
  def map[U](fn: T => U): Table[U] = macro Macros.map
}

object Macros {
  def map(c: Context)(fn: c.Tree): c.Tree = {
    val subquery: c.Tree = translate(fn)
    q"new Table(Map(${c.prefix}, $subquery))"
  }
}
```

Old macros

```
val users: Table[User] = ...
users.map(u => u.name)
```

⬇

```
val users: Table[User] = ...
new Table(Map(users, Ref("name", classOf[String])))
```

- When the user writes `Table.map`, the compiler calls `Macros.map`
- `Macros.map` expands in a domain-specific fashion
- Compiler replaces the call to `Table.map` with the macro expansion

Old macros

```
val users: Table[User] = ...
users.map(u => u.name)
```

⇓

```
val users: Table[User] = ...
new Table(Map(users, Ref("name", classOf[String])))
```

- When the user writes `Table.map`, the compiler calls `Macros.map`
- `Macros.map` expands in a domain-specific fashion
- Compiler replaces the call to `Table.map` with the macro expansion

Old macros

```
val users: Table[User] = ...
users.map(u => u.name)
```

⬇

```
val users: Table[User] = ...
new Table(Map(users, Ref("name", classOf[String])))
```

- When the user writes `Table.map`, the compiler calls `Macros.map`
- `Macros.map` expands in a domain-specific fashion
- Compiler replaces the call to `Table.map` with the macro expansion

New macros

```
class Table[T](val query: Query[T]) {
  inline def map[U](fn: T => U): Table[U] = meta {
    val subquery: c.Tree = translate(fn)
    q"new Table(Map($this, $subquery))"
  }
}
```

New macros

```
val users: Table[User] = ...
users.map(u => u.name)
```

⇩

```
val users: Table[User] = ...
meta{ ...; q"new Table(Map(users, $subquery))" }
```

⇩

```
val users: Table[User] = ...
new Table(Map(users, Ref("name", classOf[String])))
```

- When the user writes `Table.map`, the compiler inlines its rhs
- Compiler expands the `meta` block using scala.meta
- The results of the meta expansion are inlined again

New macros

```
val users: Table[User] = ...
users.map(u => u.name)
```

```
val users: Table[User] = ...
meta{ ...; q"new Table(Map(users, $subquery))" }
```

```
val users: Table[User] = ...
new Table(Map(users, Ref("name", classOf[String])))
```

- When the user writes `Table.map`, the compiler inlines its rhs
- Compiler expands the `meta` block using scala.meta
- The results of the meta expansion are inlined again

New macros

```
val users: Table[User] = ...
users.map(u => u.name)
```

⬇

```
val users: Table[User] = ...
meta{ ...; q"new Table(Map(users, $subquery))" }
```

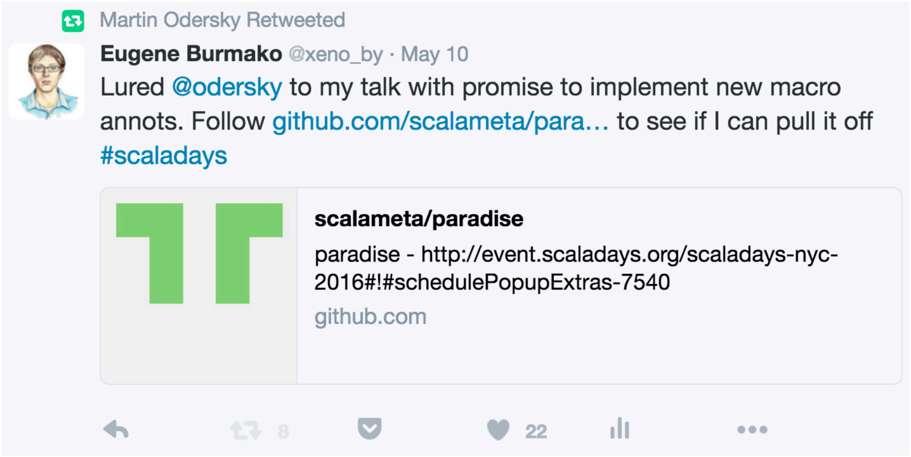
⬇

```
val users: Table[User] = ...
new Table(Map(users, Ref("name", classOf[String])))
```

- When the user writes `Table.map`, the compiler inlines its rhs
- Compiler expands the `meta` block using scala.meta
- The results of the meta expansion are inlined again

Part 4: Live demo

Challenge accepted



Martin Odersky Retweeted

Eugene Burmako @xeno_by · May 10

Lured @odersky to my talk with promise to implement new macro annots. Follow github.com/scalameta/para… to see if I can pull it off #scaladays

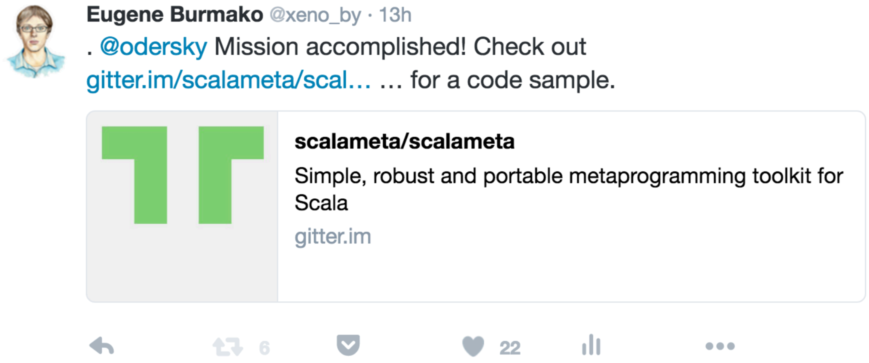
scalameta/paradise

paradise - http://event.scaladays.org/scaladays-nyc-2016#!#schedulePopupExtras-7540

github.com

8 22

Mission accomplished



Eugene Burmako @xeno_by · 13h

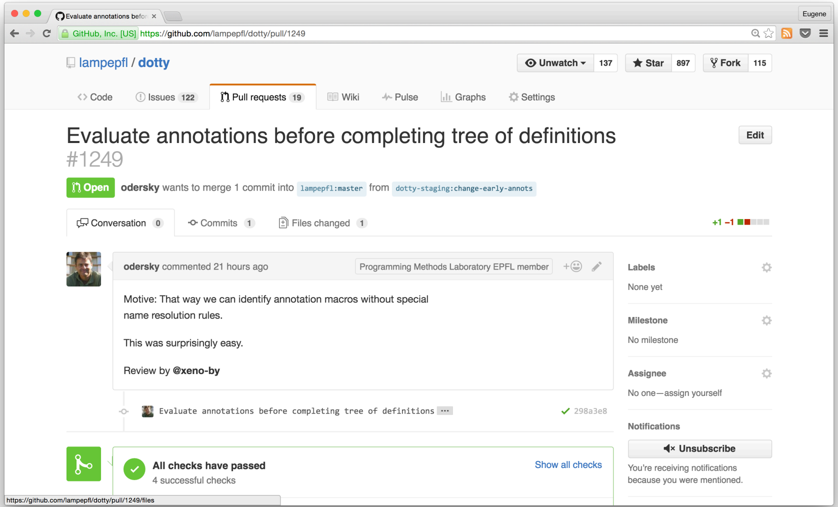
. @odersky Mission accomplished! Check out gitter.im/scalameta/scal… … for a code sample.

scalameta/scalameta

Simple, robust and portable metaprogramming toolkit for Scala

gitter.im

In the meanwhile, in the Dotty land



Evaluate annotations befor ×

GitHub, Inc. [US] https://github.com/lampepfl/dotty/pull/1249

lampepfl / dotty

Evaluate annotations before completing tree of definitions #1249

Open **odersky** wants to merge 1 commit into `lampepfl:master` from `dotty-staging:change-early-annots`

odersky commented 21 hours ago — Programming Methods Laboratory EPFL member

Motive: That way we can identify annotation macros without special name resolution rules.

This was surprisingly easy.

Review by **@xeno-by**

`Evaluate annotations before completing tree of definitions ...` ✓ 298a3e8

All checks have passed
4 successful checks
Show all checks

Labels
None yet

Milestone
No milestone

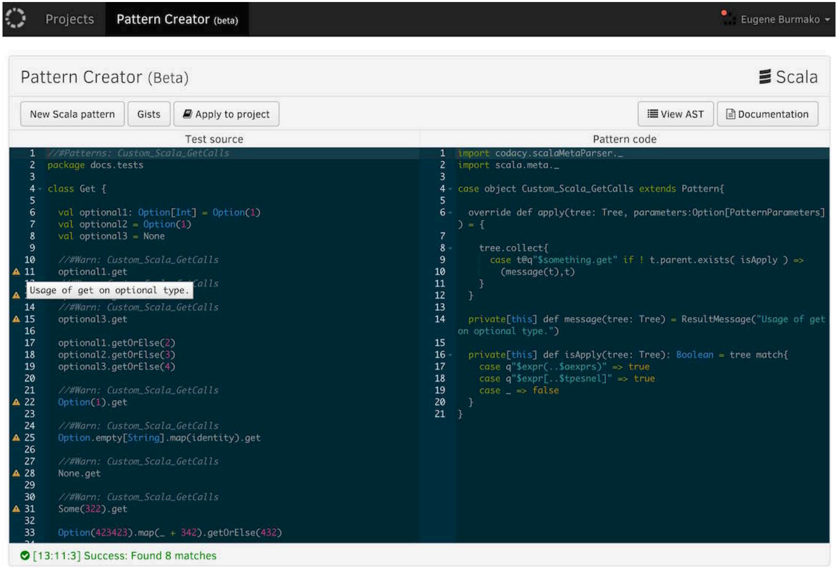
Assignee
No one—assign yourself

Notifications
Unsubscribe
You're receiving notifications because you were mentioned.

https://github.com/lampepfl/dotty/pull/1249/files

Part 5: The future

Codacy



Test source

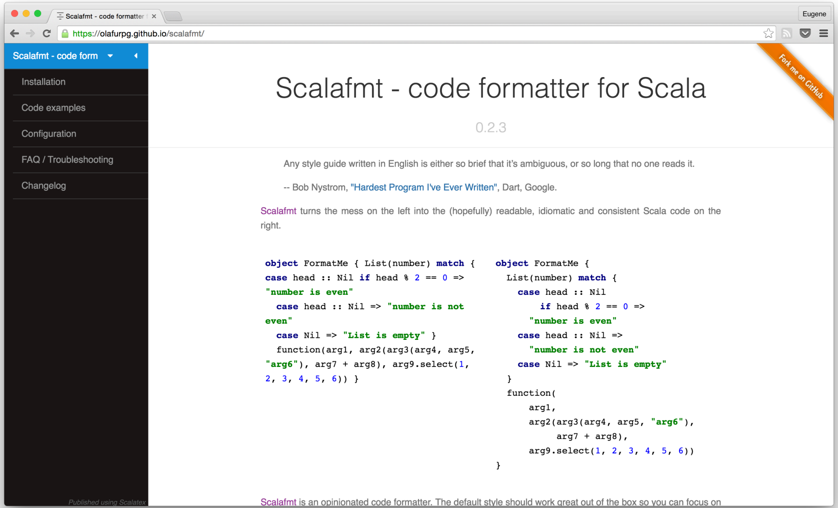
```
//#Patterns: Custom_Scala_GetCalls
package docs.tests

class Get {

  val optional1: Option[Int] = Option(1)
  val optional2 = Option(1)
  val optional3 = None

  //#Warn: Custom_Scala_GetCalls
  optional1.get
 Usage of get on optional type.  Calls

  //#Warn: Custom_Scala_GetCalls
  optional3.get

  optional1.getOrElse(2)
  optional2.getOrElse(3)
  optional3.getOrElse(4)

  //#Warn: Custom_Scala_GetCalls
  Option(1).get

  //#Warn: Custom_Scala_GetCalls
  Option.empty[String].map(identity).get

  //#Warn: Custom_Scala_GetCalls
  None.get

  //#Warn: Custom_Scala_GetCalls
  Some(322).get

  Option(423423).map(_ + 342).getOrElse(432)
```

Pattern code

```
import codacy.scalaMetaParser._
import scala.meta._

case object Custom_Scala_GetCalls extends Pattern{

  override def apply(tree: Tree, parameters:Option[PatternParameters]
) = {

    tree.collect{
      case t@q"$something.get" if ! t.parent.exists( isApply ) =>
        (message(t),t)
    }
  }

  private[this] def message(tree: Tree) = ResultMessage("Usage of get
on optional type.")

  private[this] def isApply(tree: Tree): Boolean = tree match{
    case q"$expr(..$aexprs)" => true
    case q"$expr[..$tpesnel]" => true
    case _ => false
  }
}
```

[13:11:3] Success: Found 8 matches

scalafmt



Scalafmt - code formatter for Scala

0.2.3

Any style guide written in English is either so brief that it's ambiguous, or so long that no one reads it.

-- Bob Nystrom, "Hardest Program I've Ever Written", Dart, Google.

Scalafmt turns the mess on the left into the (hopefully) readable, idiomatic and consistent Scala code on the right.

```scala
object FormatMe { List(number) match {
case head :: Nil if head % 2 == 0 =>
"number is even"
  case head :: Nil => "number is not
even"
  case Nil => "List is empty" }
  function(arg1, arg2(arg3(arg4, arg5,
"arg6"), arg7 + arg8), arg9.select(1,
2, 3, 4, 5, 6)) }
```

```scala
object FormatMe {
  List(number) match {
    case head :: Nil
        if head % 2 == 0 =>
      "number is even"
    case head :: Nil =>
      "number is not even"
    case Nil => "List is empty"
  }
  function(
      arg1,
      arg2(arg3(arg4, arg5, "arg6"),
           arg7 + arg8),
      arg9.select(1, 2, 3, 4, 5, 6))
}
```

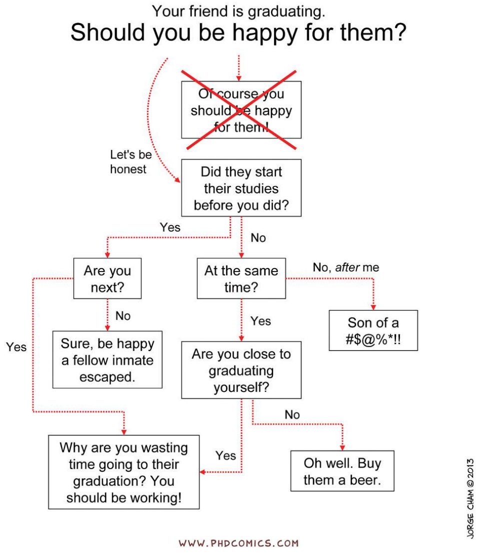
Scalafmt is an opinionated code formatter. The default style should work great out of the box so you can focus on

Inline macros

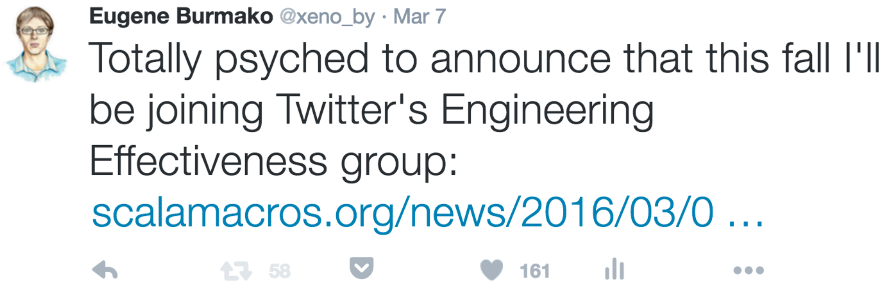
```
import scala.meta._

object main {
  inline def apply()(defn: Any) = meta {
    val q"object $name  ..$stats " = defn
    val main = q"""
      def main(args: Array[String]): Unit = { ..$stats }
    """
    q"object $name { $main }"
  }
}

@main object Test {
  println("hello world")
}
```

Dotty linker

```
import dotty.linker._

@rewrites
object Rewrites {
  def metaRule[T](x: List[T]) =
    Rewrite(from = x.toString,
                 to = meta { /* entry point to scala.meta */ })
}
```


Your friend is graduating.
Should you be happy for them?
~~Of course you should be happy for them!~~
Let's be honest
Did they start their studies before you did?
Yes
No
Are you next?
At the same time?
No, *after* me
Son of a #$@%*!!
No
Yes
Sure, be happy a fellow inmate escaped.
Yes
Are you close to graduating yourself?
No
Yes
Why are you wasting time going to their graduation? You should be working!
Oh well. Buy them a beer.
JORGE CHAM © 2013
WWW.PHDCOMICS.COM

The road ahead



Eugene Burmako @xeno_by · Mar 7

Totally psyched to announce that this fall I'll be joining Twitter's Engineering Effectiveness group: scalamacros.org/news/2016/03/0 …

58 161

Wrapping up

Summary

- We've just released our first beta version
- The project is officially endorsed and funded
- Current users: Codacy, scalafmt
- Future users: new macros, Dotty linker
- Try it out!