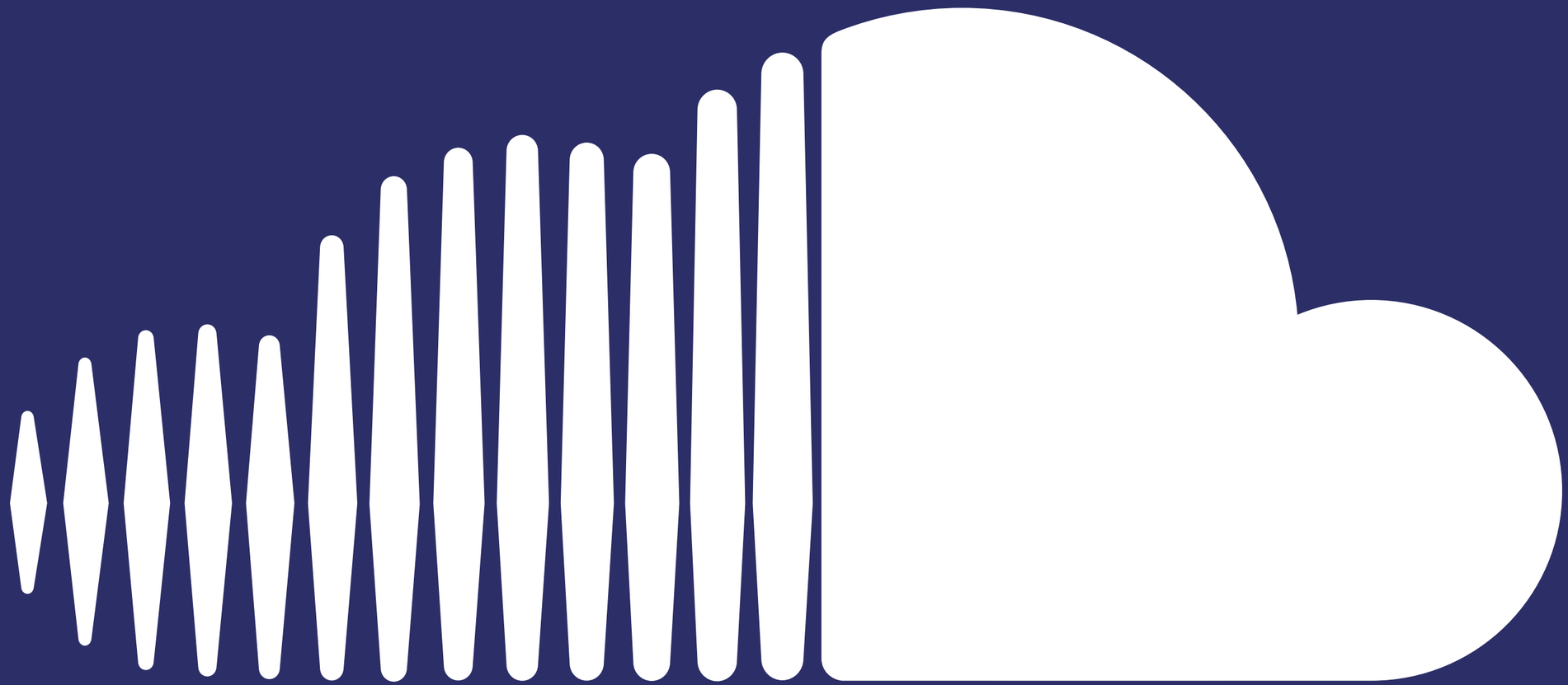
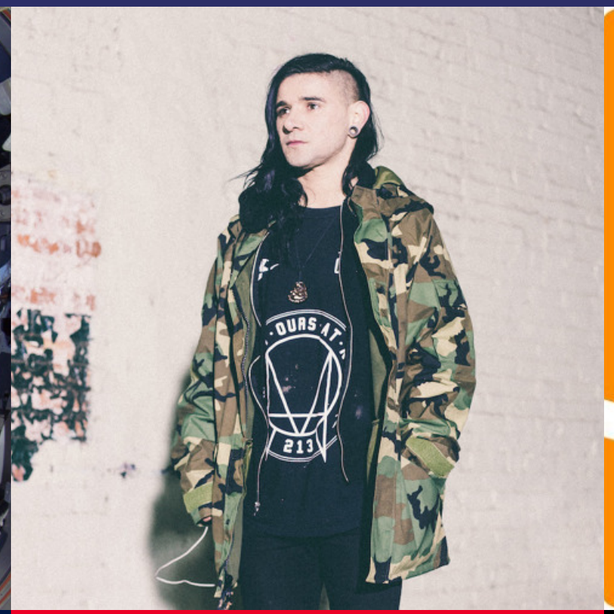
# My name is @folone

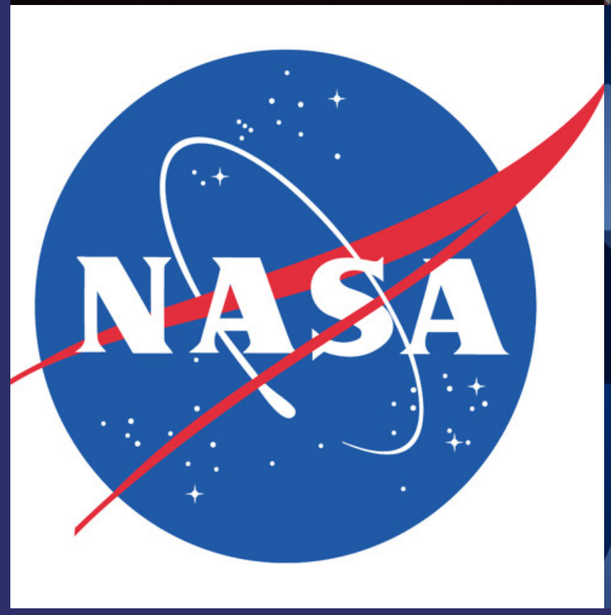
https://github.com/folone/fby



SOUNDCLOUD

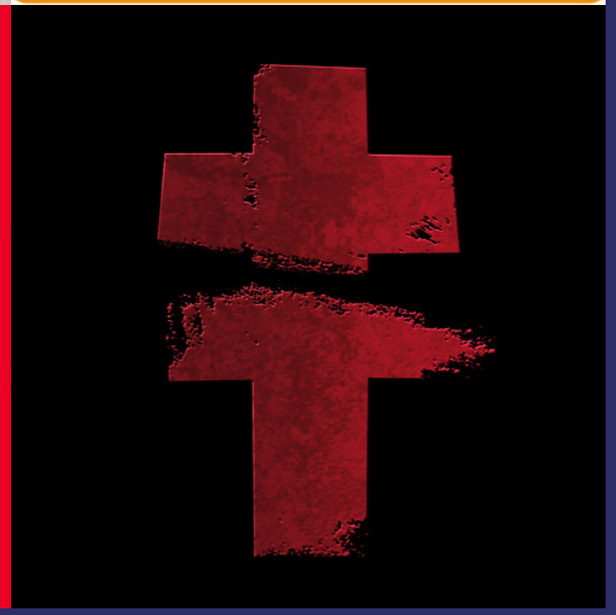

AC/DC
ROCK OR BUST




NASA


Haskell Cast


LED ZEPPELIN




→ 12 hours uploaded every minute

→ ~35k listening years every month

→ >135M tracks (including content from majors: Sony/Universal/Warner)

→ ~180M monthly active users

Search for topics

Groups

POST REPLY

Data Center & Network Engineering ›

# Maintenance postponed by rapper diss war. :-/

3 posts by 1 author

g+1

**ryan**

**Other recipients:** tkau...@batblue.com, bpa...@batblue.co


SOUNDCLOUD

# Type level programming in Scala

```
sealed trait Bool {
  type &&[B <: Bool] <: Bool
  type ||[B <: Bool] <: Bool
  type IfElse[T, F] <: Any
}
trait True extends Bool {
  type &&[B <: Bool] = B
  type ||[B <: Bool] = True
  type IfElse[T, F] = T
}
trait False extends Bool {
  type &&[B <: Bool] = False
  type ||[B <: Bool] = B
  type IfElse[T, F] = F
}
```

```
// false || true == true
implicitly[False # `||` [True] =:= True]

// if(true) String else Int
implicitly[True # IfElse[String, Int] =:= String]

/* if(true) {
 *   if(false) Long else String
 * } else Int
 */
implicitly[True # IfElse[False # IfElse[Long, String], Int] =:= String]
```

# Peano numbers!

```
trait Nat
trait Z extends Nat
trait Succ[A <: Nat] extends Nat
```

```scala
trait MinusOne[A <: Nat] {
  type Res <: Nat
}

object MinusOne {

  type Aux[A <: Nat, Res1 <: Nat] = MinusOne[A] { type Res = Res1 }

  implicit val baseCase: Aux[Z, Z] = new MinusOne[Z] {
    type Res = Z
  }
  implicit def inductiveCase[A <: Nat]: Aux[Succ[A], A] = new MinusOne[Succ[A]] {
    type Res = A
  }
}
```

```
trait Plus[A <: Nat, B <: Nat] {
  type Res <: Nat
}
object Plus {
  type Aux[A <: Nat, B <: Nat, Res1 <: Nat] = Plus[A, B] { type Res = Res1 }

  implicit def baseCase[A <: Nat]: Aux[A, Z, A] = new Plus[A, Z] {
    type Res = A
  }
  implicit def inductiveCase[A <: Nat, B <: Nat, C <: Nat, D <: Nat]
    (implicit ev0: MinusOne.Aux[B, C],
              ev1: Plus.Aux[Succ[A], C, D]): Aux[A, B, D] =
    new Plus[A, B] {
      type Res = D
    }
}
```

```
implicit def inductiveCase[A <: Nat,
                           B <: Nat,
                           C <: Nat,
                           D <: Nat]
  (implicit ev0: MinusOne.Aux[B, C],
            ev1: Plus.Aux[Succ[A], C, D]): Aux[A, B, D] =
  new Plus[A, B] {
    type Res = D
  }
```

```
type _1 = Succ[Z]
type _2 = Succ[_1]
type _3 = Succ[_2]

implicitly[Plus.Aux[_1, _2, _3]]
```


# HLists!

```
trait Nat
trait Z extends Nat
trait Succ[A <: Nat] extends Nat

trait HList
trait HNil extends HList
trait HCons[A, T <: HList] extends HList
```

```
trait Length[L <: HList] {
  type Res <: Nat
}

object Length {
  type Aux[L <: HList, Res1 <: Nat] = Length[L] { type Res = Res1 }
  implicit val baseCase: Aux[HNil.type, Z] = new Length[HNil.type] {
    type Res = Z
  }
  implicit def inductiveCase[H, T <: HList, N <: Nat]
    (implicit ev0: Length.Aux[T, N]) =
    new Length[HCons[H, T]] {
      type Res = Succ[N]
    }
}
```


Shapeless!
https://github.com/milessabin/shapeless
AIRPORT
NIGHT RUN
ZIEL
BERLINER


There's a
Prolog in your
Scala


Typelog!
https://github.com/densh/typelog


# Json


Danke!
@folone
https://soundcloud.com/jobs
https://github.com/folone/fby