Simplicity *itself*

# Groovy Intro

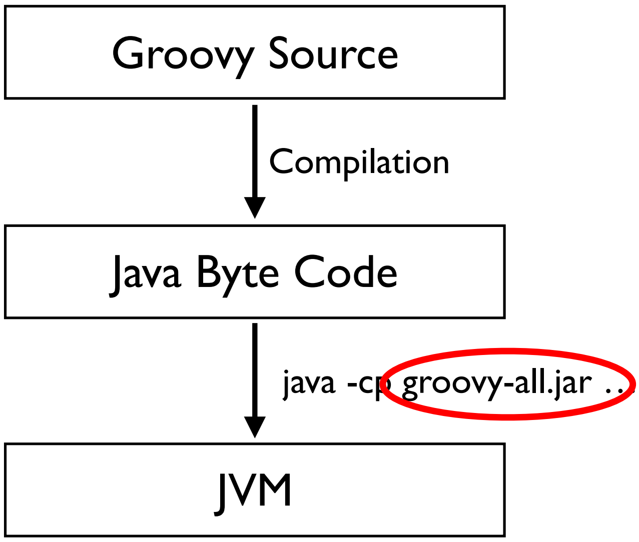
## What is Groovy?

- Object-oriented language for the JVM
- Java-like syntax
- Dynamic behaviour by default
- Optional types
- Optional compile-time type checking

# Prerequisites

Java Development Kit (JDK)

# Groovy is compiled


Groovy Source

Compilation

Java Byte Code

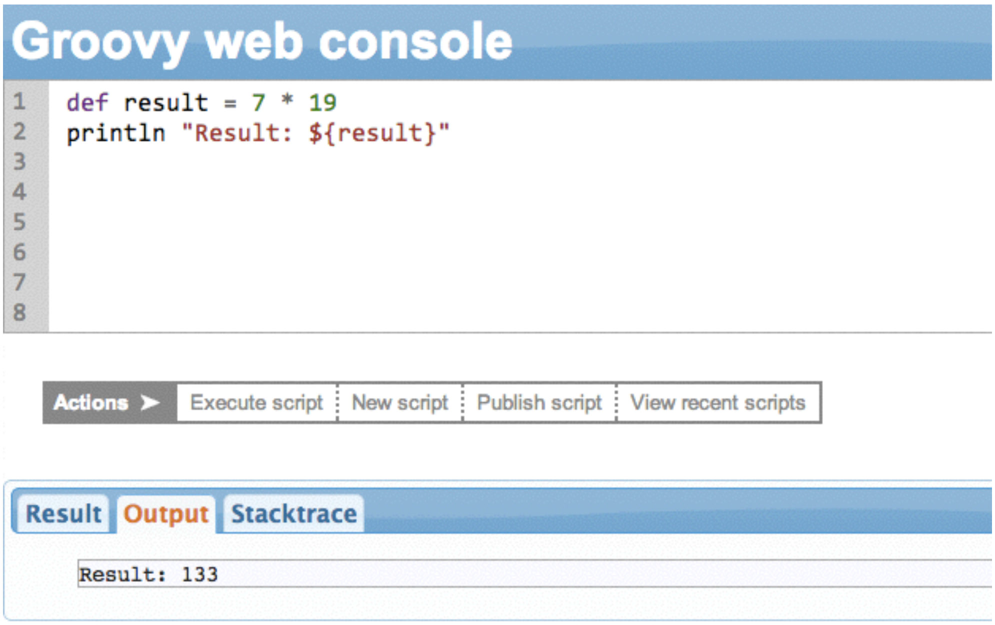
java -cp groovy-all.jar ...

JVM

# What’s it good for?

- Cross-platform scripts
- Tests
- General application development
- Embedded domain-specific languages (DSLs)

# How do you use it?

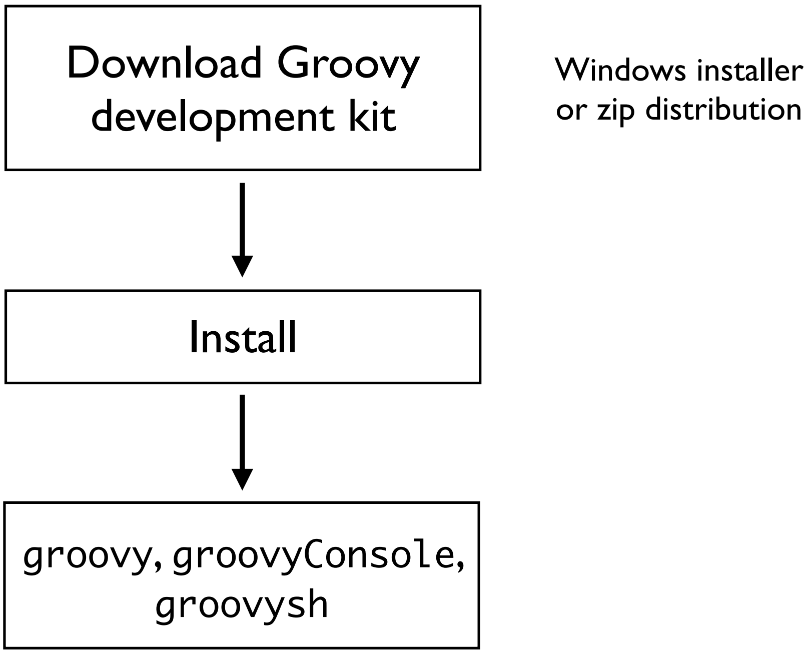


```
def result = 7 * 19
println "Result: ${result}"
```

```
Result: 133
```

http://groovyconsole.appspot.com/

# How do you use it?



Download Groovy development kit

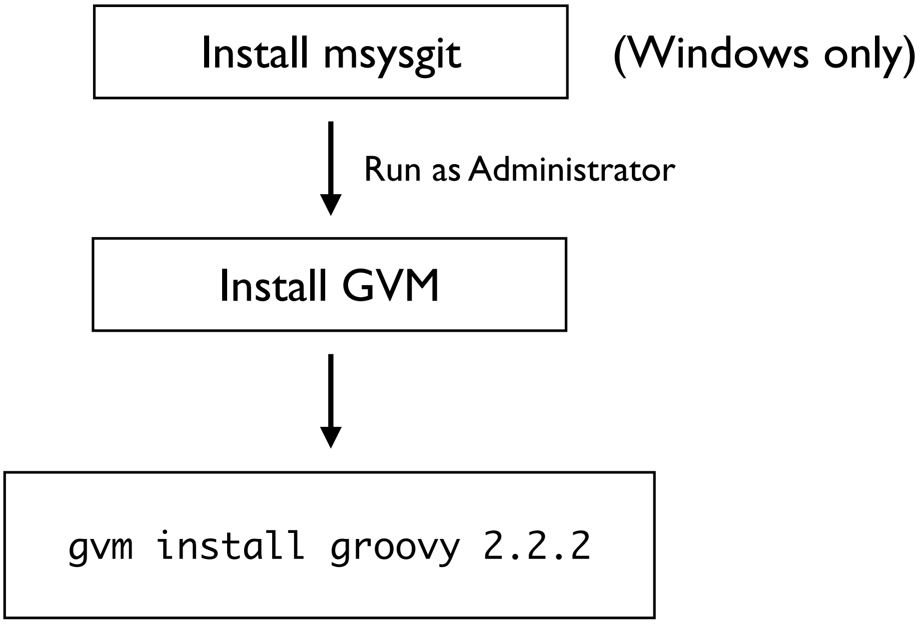
Windows installer or zip distribution

Install

`groovy`, `groovyConsole`, `groovysh`

http://groovy-lang.org/download.html

# How do you use it?



Install msysgit

(Windows only)

Run as Administrator

Install GVM

```
gvm install groovy 2.2.2
```

http://gvmtool.net/

# Tools in distribution

*groovy* Like an interpreter. Runs scripts directly from source code.

*groovyc* Compiler. Generates the class files for your Groovy source.

*groovysh* Command line REPL for Groovy.

*groovyConsole* GUI for running Groovy scripts. More flexible than *groovysh*.

# IDEs - Eclipse

Install Groovy Eclipse plugin:

http://dist.springsource.org/snapshot/GRECLIPSE/e4.5/

- Groovy version fixed for open workspace
- Changing Groovy version requires restart
- Replace '4.5' with your Eclipse version
- Groovy distribution is **not** required

# IDEs - IntelliJ IDEA

Install Groovy plugin

↓

New Groovy project

or

Add Groovy JAR to module dependencies

# IDEs - NetBeans

Activate Groovy plugin

# Build tools - Maven

Groovy Eclipse Compiler

or

GMaven Plus plugin

# Build tools - Gradle

```
apply plugin: "groovy"
```

and

```
dependencies {
    compile "org.codehaus.groovy:groovy-all:2.4.3"
}
```

# Scripts vs Classes

# A script

- Runnable Groovy code
- Can access program arguments
- Typically run direct from source

# A script

Local variable

```
def numbers = [1, 4, 8, -2, 0]
println "The squares of ${numbers}"
println "are: " + numbers.collect { it ** 2 }
```

Sequence of statements
outside of a class definition

# A class

- A custom type
- Defines properties and methods
- Only runnable if it has a *public static void main* method

# A class

A namespace

Starts a type definition

```
package org.example

class Person {
    String name
    int    age

    def greeting() {
        return "Hi ${name}"
    }
}
```

Can use declared types for object state

You don't *need* to use declared types

String interpolation

# Static vs instance

```
class Person {
    static void sayHello(String name) {
        println "Hi ${name}"
    }
}
```

↓

```
Person.sayHello("Peter")
```

# Static vs instance

```
class Person {
    void sayHello(String name) {
        println "Hi ${name}"
    }
}
```

↓

```
def p = new Person()
p.sayHello("Peter")
```

# Properties

```
class Person {
    String name
}
```

↓

```
def p = new Person(name: "Peter")
println p.name
```

# Properties

```
class Person {
    String name
}
```

```
def p = new Person(name: "Peter")
println p.getName()
```

# Properties

```
class Person {
    private String internalName

    String getName() { return internalName }
    void setName(String n) { internalName = n }
}
```

```
def p = new Person(name: "Peter")
println p.name
```

# Properties

- Defined by getters and setters
- Boolean properties can be `is<Prop>()`
- Properties can be declared like fields
  - but not with visibility scope (public, private, protected)

# Basic types

# Numbers

| | | |
|---|---|---|
| `Byte` | `byte` | 8-bit signed integer |
| `Short` | `short` | 16-bit signed integer |
| `Integer` | `int` | 32-bit signed integer |
| `Long` | `long` | 64-bit signed integer |
| `Float` | `float` | 32-bit floating point |
| `Double` | `double` | 64-bit floating point |
| `BigInteger` | - | Arbitrary-sized signed integer |
| `BigDecimal` | - | Arbitrary-sized, exact floating point |

# Objects vs primitives

- Object values can be `null`
- Primitives *may* be more performant
- Unlike Java, you can call methods on primitive values!

# Other types

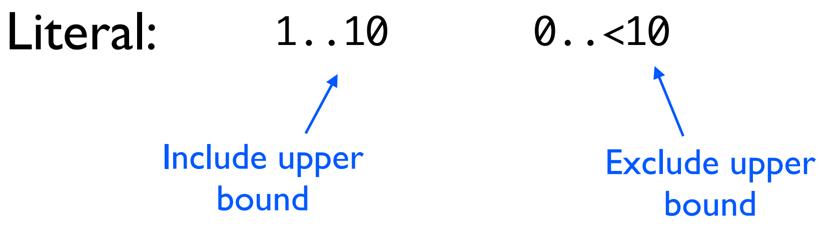
| | | |
|---|---|---|
| `String` | | Unicode text. Internally stored as UTF-16. |
| `Character` | `char` | Single, 16-bit Unicode character. |
| `Boolean` | `boolean` | True or false type. |

`java.util` package:

| | |
|---|---|
| `Date` | An instant in time (ms since 1970-01-01 00:00) |
| `List` | An ordered collection (sequence) of objects |
| `Set` | A collection of *unique* (enforced) objects |
| `Map` | A collection of key-value pairs |

# Other types

| | |
|---|---|
| `groovy.lang.Range` | Lower and upper bound representing a range of values. |



Literal: `1..10` `0..<10`

Include upper bound

Exclude upper bound

# What's my type?

```
obj.getClass()
```

Returns object of type `Class` - the `toString()` method returns the fully qualified class name

# Docs on types

If you have a type that's part of Java class library:

https://docs.oracle.com/javase/8/docs/api/

Main Java API docs

http://www.groovy-lang.org/gdk.html

Groovy's extensions to the core Java classes

# Optional types

# What?

```
def name = "Peter"
```

vs

```
String name = "Peter"
```

```
def reverse(text) { ... }
```

vs

```
String reverse(String text) { ... }
```

# Why?

- Untyped, because Groovy is a dynamic language
- Types:
  - provide better integration with Java
  - can document intent
  - are not enforced at compile time, but at runtime
- Untyped often easier to test

# What do I do?

- Use types for:
  - properties
  - method signatures
- Use `def` for
  - local variables
  - for significantly easier testing

# Potential errors

```
Date now = "test"
```

↓

org.codehaus.groovy.runtime.typehandling.GroovyCastException: Cannot cast object 'now' with class 'java.lang.String' to class 'java.util.Date'

        at ConsoleScript2.run(ConsoleScript2:1)

# Potential errors

```
Date now = "test"
```

↓

org.codehaus.groovy.runtime.typehandling.GroovyCastException: Cannot cast object 'now' with class 'java.lang.String' to class 'java.util.Date'

        at ConsoleScript2.run(ConsoleScript2:1)

# Potential errors

```groovy
def name = "test"
assert !(name.contains(123))
```

↓

```
groovy.lang.MissingMethodException: No signature of method:
    java.lang.String.contains() is applicable for argument types:
    (java.lang.Integer) values: [123]
Possible solutions: contains(java.lang.CharSequence),
contains(java.lang.CharSequence), toString(), toString(), toString(),
notify()
        at ConsoleScript0.run(ConsoleScript0:2)
```

# Potential errors

```groovy
def name = "test"
assert !(name.contains(123))
```

↓

```
groovy.lang.MissingMethodException: No signature of method:
    java.lang.String.contains() is applicable for argument types:
    (java.lang.Integer) values: [123]
Possible solutions: contains(java.lang.CharSequence),
contains(java.lang.CharSequence), toString(), toString(), toString(),
notify()
        at ConsoleScript0.run(ConsoleScript0:2)
```

Copyright © 2015 Cacoethes Software

# What happens with…

```
String number = 100
```

- No compilation error
- No runtime error

➡ Integer is *implicitly coerced* to a string

- Implicit coercion only happens for local variables and fields/properties
- Coercion must be known by Groovy

# Operators

# Operator overloading

```
1 + 1
```

is equivalent to

```
1.plus(1)
```

See Groovy JDK for `Number.plus()`

# Operator overloading

- Unsupported operators appear as MissingMethodExceptions
- Operator => method mapping in Groovy Syntax Quick Reference
- Implement the methods in your own classes to support the corresponding operators

# APIs

# Filesystem

`java.io` package:

| | |
|---|---|
| `File` | File path with `exists()`, `isFile()`, etc. |
| `InputStream` | For reading binary data |
| `OutputStream` | For writing binary data |
| `Reader` | For reading text |
| `Writer` | For writing text |

# Filesystem (Java 7+)

`java.nio.file` package:

| | |
|---|---|
| `Path` | Abstract file path |
| `FileSystem` | Represents a filesystem. Mainly used to create `Path`s. |
| `FileSystems` | Factory methods for creating `FileSystem` objects. |
| `Files` | Static methods to perform file and directory operations. |

# Network

`java.net` package:

| | |
|---|---|
| `URI` | Preferred URI type |
| `URL` | Legacy URI type |
| `URLConnection` | For reading and writing data to a URL |

3rd party libraries preferred!

# Java 8+ Date & Time

`java.time` package:

| | |
|---|---|
| `Instant` | A fixed point in time. |
| `Duration` | A period of time. |
| `LocalDateTime` | Date and time relative to the computer it's running on. Cannot represent an `Instant`. |
| `OffsetDateTime` | Date and time relative to UTC. *Can* represent an `Instant`. |

Use Joda Time on Java 7 & earlier

# External processes

`java.lang` package:

| | |
|---|---|
| `Runtime` | Provides access to the JVM itself. |
| `ProcessBuilder` | Spawns external processes. |

# Libraries

- Distributed as JARs (Java ARchives)
- Zips containing compiled classes
- Unmanaged by JVM
- No builtin dependencies between JARs
- Added to the JVM's "classpath"

# Diagnosing Problems

# NullPointerException

```
class Text {
    def content

    def reverse() {
        return content.reverse()
    }

    def processText() {
        return reverse()
    }
}

def text = new Text()
println text.processText()
```

# NullPointerException

```
java.lang.NullPointerException: Cannot invoke
method reverse() on null object
   at Text.reverse(ConsoleScript4:5)
   at Text.processText(ConsoleScript4:9)
   at Text$processText.call(Unknown Source)
   at ConsoleScript4.run(ConsoleScript4:14)
```

# NullPointerException

```
java.lang.NullPointerException: Cannot invoke
method reverse() on null object
   at Text.reverse(ConsoleScript4:5)
   at Text.processText(ConsoleScript4:9)
   at Text$processText.call(Unknown Source)
   at ConsoleScript4.run(ConsoleScript4:14)
```

# NullPointerException

```
java.lang.NullPointerException: Cannot invoke
method reverse() on null object
   at Text.reverse(ConsoleScript4:5)
   at Text.processText(ConsoleScript4:9)
   at Text$processText.call(Unknown Source)
   at ConsoleScript4.run(ConsoleScript4:14)
```

# NullPointerException

```
java.lang.NullPointerException: Cannot invoke
method reverse() on null object
   at Text.reverse(ConsoleScript4:5)
   at Text.processText(ConsoleScript4:9)
   at Text$processText.call(Unknown Source)
   at ConsoleScript4.run(ConsoleScript4:14)
```

# NullPointerException

```
java.lang.NullPointerException: Cannot invoke
method reverse() on null object
   at Text.reverse(ConsoleScript4:5)
   at Text.processText(ConsoleScript4:9)
   at Text$processText.call(Unknown Source)
   at ConsoleScript4.run(ConsoleScript4:14)
```

# MissingMethodException

```
class Text {
    def content

    def reverse() {
        return content.revers()
    }

    def processText() {
        return reverse()
    }
}

def text = new Text(content: "abcs")
println text.processText()
```

# MissingMethodException

```
groovy.lang.MissingMethodException: No signature
of method: java.lang.String.revers() is
applicable for argument types: () values: []
Possible solutions: reverse(), every(),
every(groovy.lang.Closure), intern(), previous(),
center(java.lang.Number)
   at Text.reverse(ConsoleScript6:5)
   at Text.processText(ConsoleScript6:9)
   at Text$processText.call(Unknown Source)
   at ConsoleScript6.run(ConsoleScript6:14)
```

# MissingPropertyException

```
class Text {
    def content

    def reverse() {
        return content.reverse
    }

    def processText() {
        return reverse()
    }
}

def text = new Text(content: "abcs")
println text.processText()
```

# MissingPropertyException

```
groovy.lang.MissingPropertyException: No such
property: reverse for class: java.lang.String
   at Text.reverse(ConsoleScript7:5)
   at Text.processText(ConsoleScript7:9)
   at Text$processText.call(Unknown Source)
   at ConsoleScript7.run(ConsoleScript7:14)
```

# Compilation error

```
class Text {
    def content

    def reverse() {
        return content.reverse
    }

    def processText() {
        return reverse()
    }
}

def text = new Text(content: "abcs"
println text.processText()
```

# Compilation error

```
1 compilation error:

unexpected token: println at line: 14, column: 1
```

Without static typing, compilation errors are almost always about bad syntax

# Resources

Java API docs:

https://docs.oracle.com/javase/8/docs/api/

Groovy JDK:

http://www.groovy-lang.org/gdk.html

Groovy API docs:

http://www.groovy-lang.org/api.html

Groovy syntax quick reference:

http://pledbrook.github.io/groovy-cheat-sheet/guide/index.html

Practical Groovy:

http://pledbrook.github.io/practical-groovy-public/

Groovy Goodness:

http://mrhaki.blogspot.co.uk/search/label/Groovy%3AGoodness