CZJUG 2009

Gradle

Hans Dockter

Gradle Project Manager

hans@gradle.biz

About Me

- Founder and Project Lead of Gradle

- CEO of Gradle Inc.

- In the old days: Committer to JBoss (Founder of JBoss-IDE)

Gradle Background

- Very active community (mailing-list, patches, issues, ...)

- Apache v2 license.

- Excellent Documentation (200+ pages user's guide)

- Frequent releases (Every 2 months)

- Quality is king:

  - 2800 unit tests, Many integration test

  - Healthy codebase

- A Groovy DSL but a Java core

- Commiter -> Steve Appling, Hans Dockter, Tom Eyckmans, Adam Murdoch, Russel Winder

Live Demo Java Project

Dependency Based Programming

```groovy
task hello << {
    println 'Hello world!'
}


task intro(dependsOn: hello) << {
    println "I'm Gradle"
}


4.times { counter ->
    task "task_$counter" << {
        println "I'm task $counter"
    }
}
```

```
> gradle hello task1
Hello world!
I'm task 1
```

Rich API

```
task hello
println hello.name
hello.dependsOn distZip
hello << { println 'Hello' }

task distZip(type: Zip)
distZip.fileSet(dir: 'somePath')
distZip.baseName = hello.name
```

Very Rich API: Test Task

- Register listeners for test execution (this works even in forked mode)

    - Get informed about a started test execution

    - Get informed about a failing or succeeding test

- Provides an API to ask for execution results.

- Part of Gradle 0.9

One way configuration Ant

```
<target name="test" depends="compile-test">
    <junit>
      <classpath refid="classpath.test" />
      <formatter type="brief" usefile="false" />
      <test name="${test.suite}"/>
    </junit>
</target>
```

One way configuration Maven

```xml
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.4.2</version>
    <configuration>
        <includes>
            <include>Sample.java</include>
        </includes>
    </configuration>
</plugin>
```

Plugins

- Java, Groovy, Scala (including mixed compilation)

- War, Jetty

- Code-Quality (Checkstyle, Codenarc)

- OSGi

- Maven

- Eclipse project file generation

Custom Plugins

```
public class CustomPlugin implements Plugin {
    public void use(final Project project, ProjectPluginsContainer plugins) {
        project.myProperty = 'myValue'
        plugins.withType(JavaPlugin).allPlugins {
            project.getTasks().add('someName', SomeType.class)
        }
        plugins.withType(WarPlugin.class).allPlugins {
            // do something with the project
        }
    }
}
```

Deep API

```
tasks.whenTaskAdded { task ->
    task.description = 'This is a new task'
}


tasks.allTasks { task ->
    task.description = 'I am new or old'
}


tasks.withType(Zip).whenTaskAdded { task ->
    task.fileSet(dir: 'someDir')
}


liveLibs = tasks.withType(Jar)
task someJar(type: Jar)
println liveLibs.all // prints ['someJar']
```

Rules

```
tasks.addRule("Pattern: ping<ID>") { String taskName ->
    if (taskName.startsWith("ping")) {
        task(taskName) << { // add task
            println "Pinging: " + (taskName - 'ping')
        }
    }
}
task groupPing {
    dependsOn pingServer1, pingServer2
}
```

```
> gradle pingServer545
Server545
```

```
> gradle groupPing
Server1
Server2
```

Dependency Management 1

```
usePlugin('java')
...
configurations {
    allJar.extends runtime
}

dependencies {
    compile "commons-lang:commons-lang:3.1", "org.hibernate:hibernate:3.2"
    runtime "mysql:mysql-connector-java:5.1.6"
    testCompile "junit:junit:4.4"
    allJar "commons-io:commons-io:1.4"
}
...
task jarAll(type: Jar) {
    merge(dependencies.allJar.resolve())
}
```

Dependency Management 2

```
usePlugin('java')
...

dependencies {
    compile "commons-lang:commons-lang:3.1", "org.hibernate:hibernate:3.2"
    runtime "mysql:mysql-connector-java:5.1.6"
    testCompile "junit:junit:4.4"
}
...
task showDeps << {
    println(configurations.runtime.files { dep -> dep.group == 'myGroup' })
}
```

Dependency Management 3

- Excludes per configuration or dependency

- You can define rules for configuration and dependencies (as for tasks)

- Very flexible repository handling

- Retrieve and deploy from/to Maven repositories

- Dependencies can have dynamic properties

- And much more

Using Ant Tasks

```
ant {
    taskdef name: "groovyc", classname: "org.groovy.ant.Groovyc"
    groovyc srcdir: "src", destdir: "${webinf}/classes", {
        classpath {
            fileset dir: "lib" {
                include name: "*.jar"
            }
            pathelement path: "classes"
        }
        javac source: "1.5", target: "1.5", debug: "on"
    }
}
```

Deep Integration with Ant Builds

```xml
<project>
    <target name="hello" depends="intro">
        <echo>Hello, from Ant</echo>
    </target>
</project>
```

```groovy
ant.importBuild 'build.xml'

hello.doFirst { println 'Here comes Ant' }
task intro << { println 'Hello, from Gradle'}
```

```
> gradle hello
Hello, from Gradle...Here comes Ant...[ant:echo] Hello, from Ant
```

Smart and Configurable Execution

Deep API

```
usePlugin('java')
...
compile.doFirst { compileTask ->
  println build.taskGraph.allTasks
  if (build.taskGraph.hasTask('codeGeneration')) {
    compileTask.exclude 'com.mycomp.somePackage'
  }
}


build.taskGraph.beforeTask { task ->
  println "I am executing now $task.name"
}


build.taskGraph.afterTask { task, exception ->
  if (task instanceof Jetty && exception != null) {
    // do something
  }
}
```
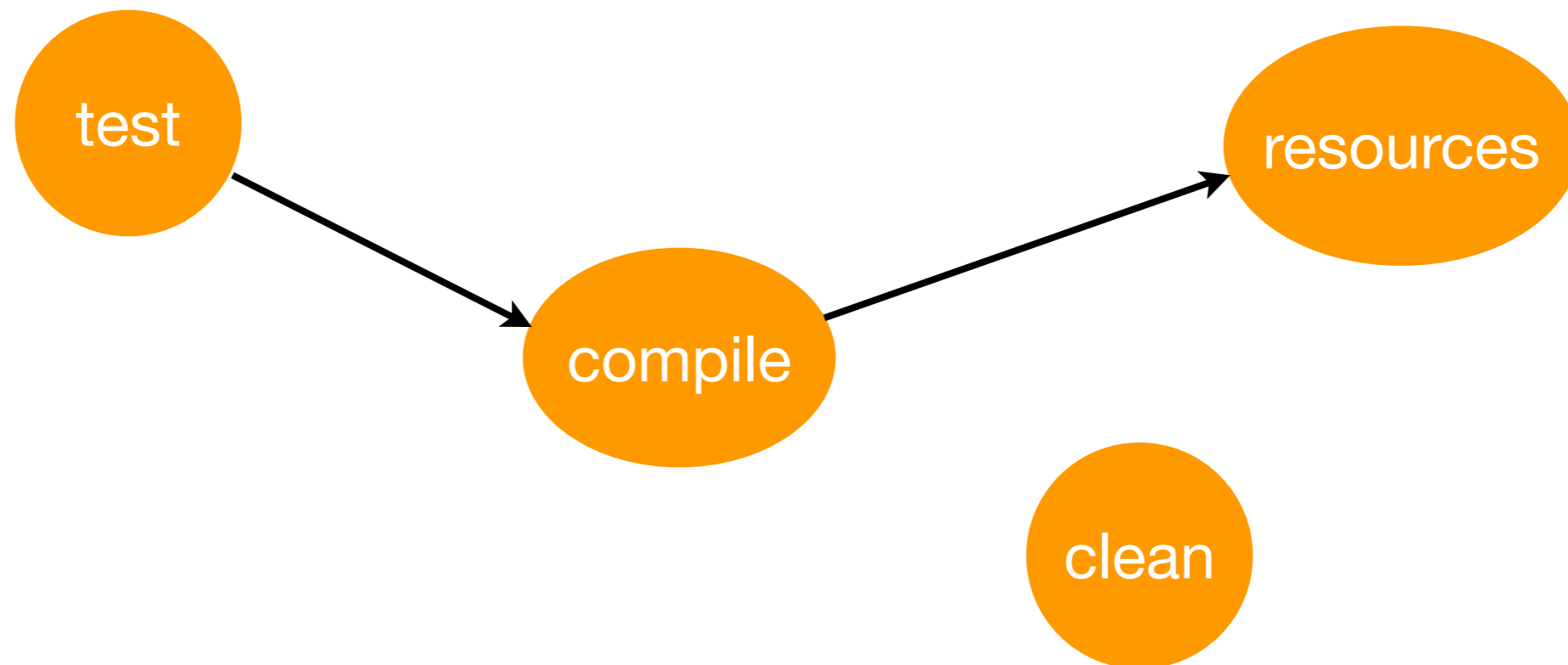
Smart Task Exclusion
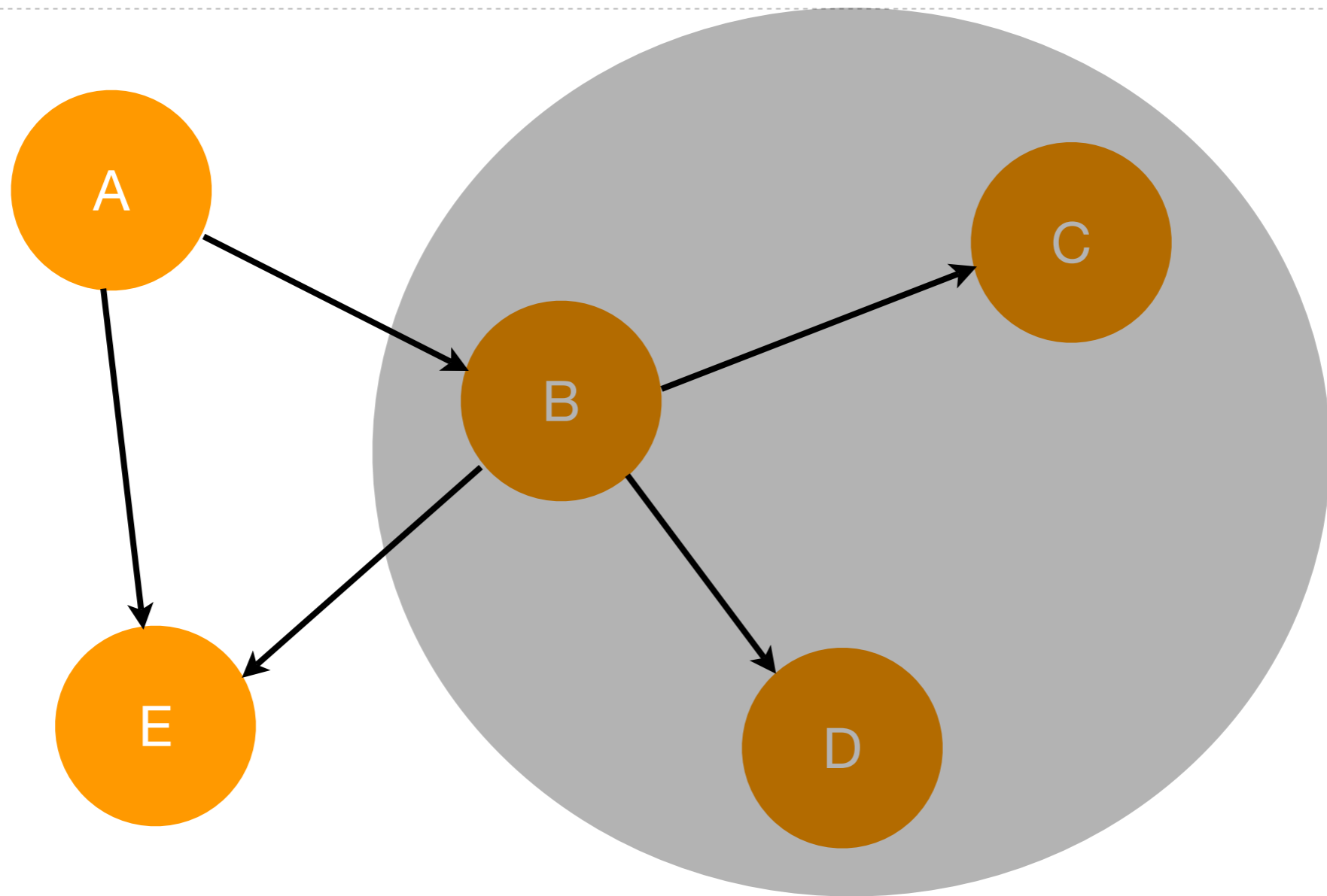
gradle A -x B executes: A, E

Smart Merging


```
> gradle clean compile test
```

```
> gradle clean compile; gradle test
```

Camel Case Execution

```
task myNameIsPrettyLong << {
    println 'Long name'
}


task someOtherTask(dependsOn: myNameIsPrettyLong) << {
    println "Other task"
}
```

```
> gradle sOtT -x mNIP
Other task
```

Smart Skip & Incremental Work

- A reliable build without **clean**.

- Only do what is necessary (e.g. incremental compile).

- We will have a generic approach that also works for custom tasks.

Multi-Core and Distributed Builds

- Gradle 0.9 comes with multi-threaded test execution.

- After that:

  - Distributed test execution.

  - Multi-Threaded Build Execution

  - Distributed Build Execution

Multi-Project Builds

- Arbitrary Multiproject Layout

- Configuration Injection

- Separate Config/Execution Hierarchy

- Partial builds

Configuration Injection

ultimateApp
   api
   webservice
   shared

```
subprojects {
  usePlugin('java')
  dependencies {
    compile "commons-lang:commons-lang:3.1"
    testCompile "junit:junit:4.4"
  }
  test {
    exclude '**/Abstract*'
  }
}
```

Separation of Config/Exec

ultimateApp
    api
    webservice
    shared

```
dependsOnChildren()
subprojects {
  usePlugin('java')
  dependencies {
    compile "commons-lang:commons-lang:3.1"
    testCompile "junit:junit:4.4"
  }
}

task dist(type: Zip) << {
  subprojects.each { subproject ->
    files(subproject.jar.archivePath)
  }
}
```

Dependencies and Partial Builds

ultimateApp
  api
  webservice
  shared

```
dependencies {
    compile "commons-lang:commons-lang:3.1", project(':shared')
}
```

Support for build, buildNeeded, buildDependent

Maven

Convention **instead of** Configuration

Frameworkitis ...

... is the disease that a framework wants to do too much for you or it does it in a way that you don't want but you can't change it. It's fun to get all this functionality for free, but it hurts when the free functionality gets in the way. But you are now tied into the framework. To get the desired behavior you start to fight against the framework. And at this point you often start to lose, because it's difficult to bend the framework in a direction it didn't anticipate. Toolkits do not attempt to take control for you and they therefore do not suffer from frameworkitis.

(Erich Gamma)

Solution

Because the bigger the framework becomes, the greater the chances that it will want to do too much, the bigger the learning curves become, and the more difficult it becomes to maintain it. If you really want to take the risk of doing frameworks, you want to have small and focused frameworks that you can also probably make optional. If you really want to, you can use the framework, but you can also use the toolkit. That's a good position that avoids this frameworkitis problem, where you get really frustrated because you have to use the framework. Ideally I'd like to have a toolbox of smaller frameworks where I can pick and choose, so that I can pay the framework costs as I go.(Erich Gamma)
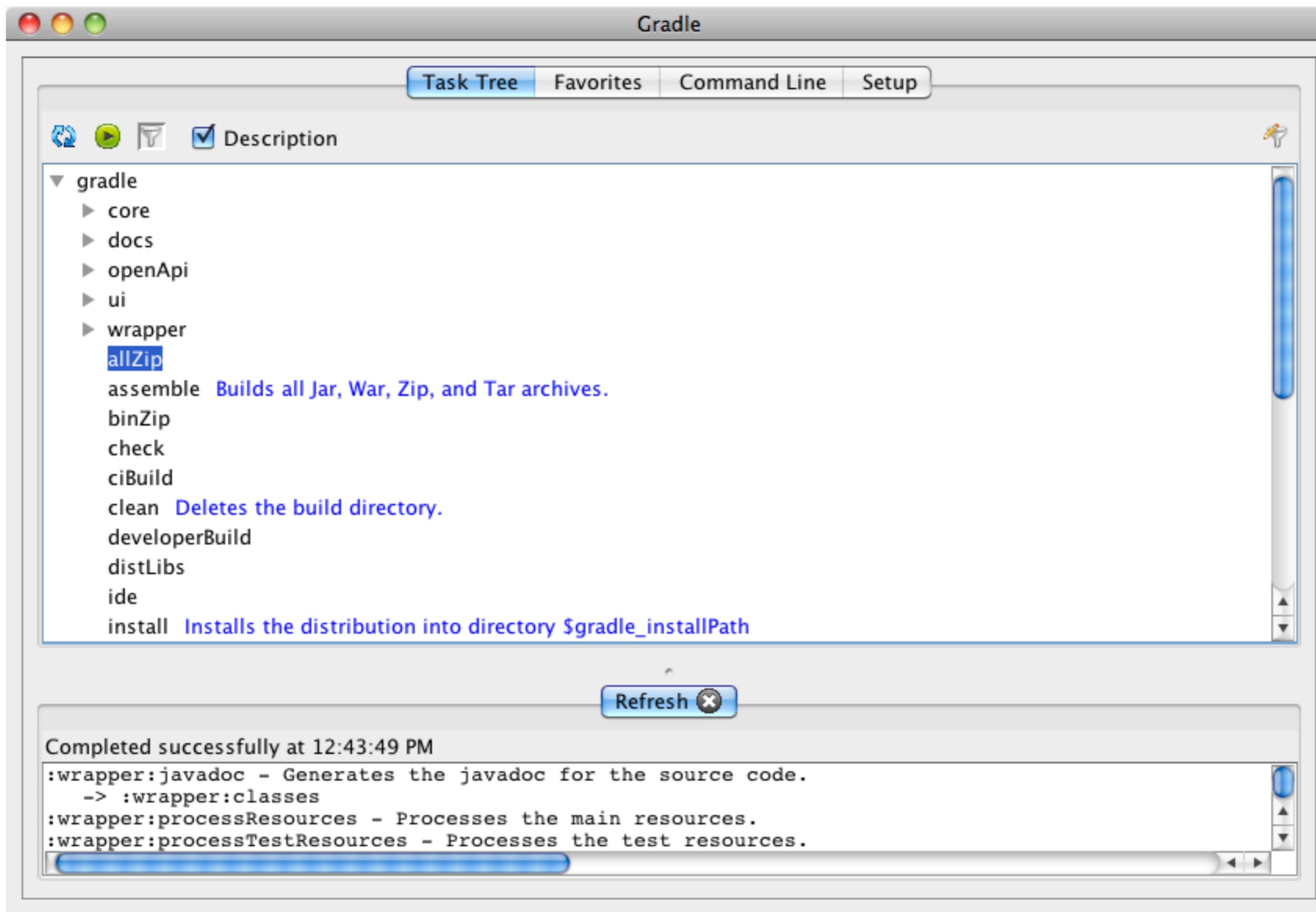
Build Language
instead of
Build Framework

Organizing Build Logic

No unnecessary indirections

If build specific:

- Within the script

- Build Sources

Otherwise: Jar

```
> gradle --gui
```

Gradle Wrapper

- Use Gradle without having Gradle installed

- Useful for CI and open source projects

Gradle 0.9

Questions

There
are
no
simple builds

Project Automation

- A build can do far more than just building the jar

- Often repetitive, time consuming, boring stuff is still done manually

    - Many of those tasks are very company specific

    - Maven & Ant are often not well suited for this

The Gradle Build

- Gradle is build with Gradle

- Automatic release management

- Automatic user's guide generation

- Automatic distribution

- Behavior depends on task execution graph

Release Management

- The version number is automatically calculated

- The distribution is build and uploaded to codehaus

- For trunk releases, a new svn branch is created.

- A tag is created.

- A new version properties file is commited.

- The download links on the website are updated

User's Guide

- The user's guide is written in DocBook and generated by our build

- The source code examples are mostly real tests and are automatically included

- The expected output of those tests is automatically included.

- The tests are run

- The current version is added to the title

Uploading & Execution Graph

Based on the task graph we set:

- Upload Destination

- Version Number