Gradle

gradleware

A Better Way to Build

What you will learn

- Core Types (Task, Plugin, SourceSets, ...)
- Build-in tasks and plugins
- Dependency Management
- Ant/Maven Integration
- Multi-Project builds

You

- Your background
  - What do you want to learn from this course?
  - What Groovy/Gradle experience do you have?
  - What build systems are you using or have experience with.
- Course Prerequisites
  - Good experience with Java
  - Groovy knowledge is helpful but not a prerequisite.
- Understanding of Ant and Maven is helpful for certain sections of this course but not a prerequisite.

Intro

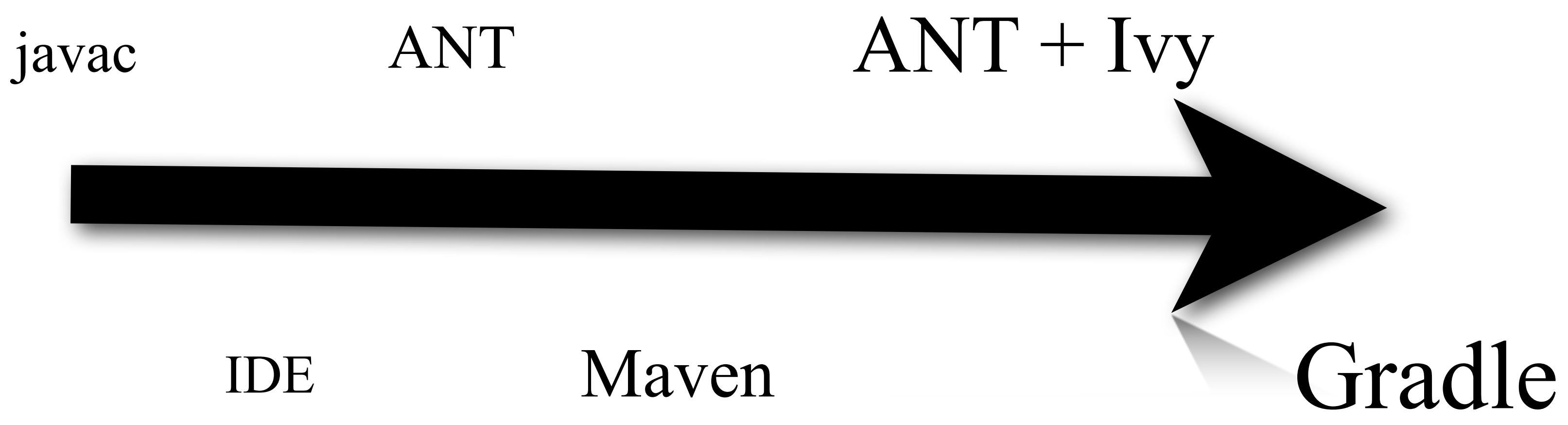
What is Gradle

- A general purpose build system
- Groovy DSL with a Java core.
- Provides build-in support for Java, Groovy, Scala, Web, OSGi.
- Exciting solutions for many of the big pain points you often have with current build systems.

Gradle Project Background

- Very active community (mailing-list, patches, issues)
- Apache v2 license.
- Excellent user's guide (200+ pages) + many samples
- Frequent releases, multiple commits per day
- Quality is king:
  - 3000 unit tests, Many hundreds of integration test
  - Healthy codebase
  - low defect rate
- Commiter -> Steve Appling, Hans Dockter, Tom Eyckmans, Adam Murdoch, Russel Winder

Java Build Tools



	APACHE ANT	Built by: maven	gradle
Basics			
Multi-language support	■■■	■■■	■■■
Dependency management	■	■■■	■■■
Versioning	■■■	■■■	■■■
Incremental builds[1]			■■■
Built-in multi-artifact builds	■■■		■■■
Multi-project dependency support[1]			■■■
Commercial support		■■■	■■■
Quality			
Unit test execution	■■■	■■■	■■■
Parallel test execution		■■■	■■■
Custom fork frequency			■■■
Custom test listeners		■	■■■
Integrates with Checkstyles, Findbugs	■■	■■■	■■■
Integrates with Sonar		■■■	Coming soon
IDE Support			
Generates eclipse workspace files	■	■■■	■■■
Customize workspace file generation		■	■■■
Runnable from Eclipse / Intellij	■■■	■■■	■■■
Auto-Import build file into IDE project	■■	■■■	Coming soon
IDE view panels[1]	■■■	■■■	Coming soon
Eco-system Integration			
CI tools (Hudson, Jenkins, Teamcity, Bamboo)	■■■	■■■	■■■
Ivy Repository	■		■■■
Maven Repository	■	■■■	■■■
Artifactory	■■■	■■■	■■■
Nexus		■■■	■■■
Import ANT builds[1]	N/A		■■■
Import Maven builds[1]		N/A	■■■
Maintainability			
Concise build script DSL			■■■
Build by convention		■■■	■■■
Auto Install / Update (zero-admin)			■■■
Extensibility			
Plugin support	■■■	■■■	■■■
In-script build extensibility			■■■
In-script programmatic control	■	■	■■■
Extensible build language[1]			■■■
Life-cycle hooks			■■■
Smart Exclusion			■■■
Customizable life-cycle			■■■

Workshop Setup

- Setup Gradle
  - Extract gradle-x.zip to a tools directory
  - Add an env var GRADLE_HOME pointing to the extracted dir
  - Add $GRADLE_HOME/bin to the PATH env var
- Setup Groovy
  - Extract groovy-x.zip to a tools directory
  - Add an env var GROOVY_HOME pointing to the extract dir
  - Add $GROOVY_HOME/bin to the PATH env var

Labs

- Lab 01 - Set Up
- Lab 02 - Quickstart

Tasks

Tasks

- Tasks are the basic unit of work in Gradle.
- Tasks have a list of actions to be executed

```
// A task with one action
task someTask << {
  // do something
}
```

```
// A task with one action
project.tasks.add('someTask').doFirst {
  // do something
}
```

DSL Syntax And Tasks

```
task hello << { println 'Hello' }
// direct API access is fine for single statements
hello.dependsOn otherTask
// for multiple access we prefer closure syntax
hello {
  onlyIf { day == 'monday' }
  dependsOn otherTask
}
// combining Configuration and Actions
task hello {
  onlyIf {
      day == 'monday'
  }
  doFirst {println 'Hello'}
}
```

Task Types and API

- Tasks have a type and API
- Default type is DefaultTask
- All tasks implement the Task interface
- Many build-in task types.
- Non-Default types usually have default action.

Task Types and API

Type: `DefaultTask`

```
task hello << { println 'Hello' }

hello.onlyIf { day == 'monday' }

task copy(type: Copy) {
  from 'someDir'
}

task whatAmIDoing
```

`Task` API

Has default action

`Copy` API

What happens in this line?

Custom Task Types

- extend DefaultTask
- Actions: @org.gradle.api.tasks.TaskAction

```groovy
class FtpTask extends DefaultTask {
  String host = 'docs.mycompany.com'
  @TaskAction
  def ftp() {
    println host
    // do something complicated
  }
}
```

Task Dependencies

- Tasks can depend on each other.
- Execution of one task requires the execution of another task first.
- Executed tasks form a directed acyclic graph.

```
// multiple ways to declare task dependencies
task foo(dependsOn: bar)
foo { dependsOn bar }
foo.dependsOn bar

// What happens here?
task foo << { dependsOn bar }
```

Using a Custom Task

```
task zip(type: Zip) {
    from jar.outputs.files
    from('scripts/') {
        fileMode = 0755
        include '**/*.sh'
        include '**/*.bat'
    }
    from('lib/') {
        include '**/*.jar'
        into('lib')
    }
    from('.') {
        include 'project.config'
    }
}
```

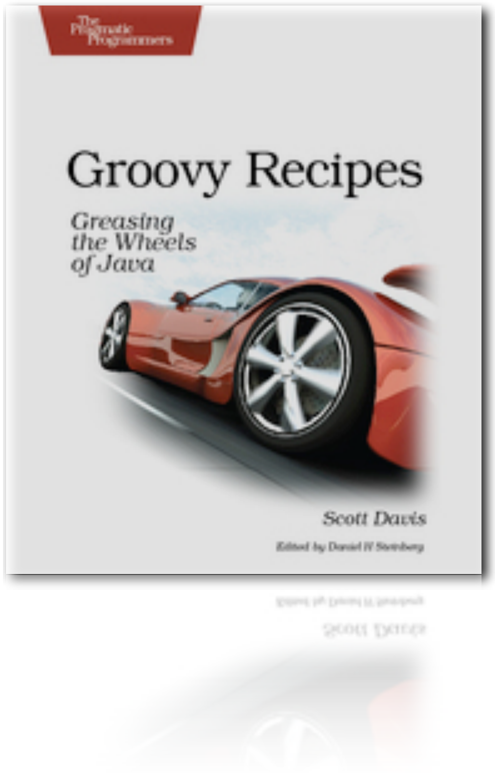
Labs

- Lab 03 - Tasks
- Lab 04 - Task Dependencies

Groovy Basics

Groovy

- A Ruby or Python like language that is tightly integrated with the Java platform.
- Compiles to byte code
- Design goal is to be easily picked up by Java devs.
- Reuse of Java semantics and API.
- Great for the creation of DSL


The Pragmatic Programmers
Groovy Recipes
Greasing the Wheels of Java
Scott Davis
Edited by Daniel H Steinberg

Groovy differences from Java

- Automatic Imports
  - java.lang.*, java.util.*, java.net.*, java.io.*, java.math.BigDecimal, java.math.BigInteger
- Optional Semicolons
  - DSL friendly
- Optional Parentheses
  - DSL friendly
- Optional Return Statements
  - last line of a method is always returned
- Optional typing
- Optional Exception Handling
- Operator Overloading
- Safe Dereferencing

Groovy and Java Classes

- Java has a lot of template "noise"
- How does Groovy simplify this class?

```
// java code
import java.util.Date;

public class Foo {

    public static void main(String[] args) {
        System.out.println(new Date());
    }
}
```

Groovy and Java Classes

- Java has a lot of template “noise”
- How does Groovy simplify this class?

```
// groovy code
println (new Date())
```

Groovy and Java Properties

- Java classes with properties are also “noisy”

```
// java code
public class Person {
    private String firstName;

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```

Groovy and Java Properties

- Groovy has sensible defaults
  - getter and setter created for us by the compiler
  - public is the default for the class
  - private is the default for firstName

```
// groovy code
class Person {
    String firstName
}
```

Groovy Closures

- Closure are like methods with a context, you can pass around.
- The Gradle DSL uses them extensively
- The Groovy API uses them extensively

```
void foo(String name, Closure cl) {
  println cl.call (name)
}

// prints gradla
foo("gradle") { s ->
  s.replace ("a", "e")
}
```

Access to Properties

- Properties can be access without using the getter method
- Under the covers the getter method is invoked

```
def s = "Gradle"

s.class.methods.each {
   println it
}
```

* If you are working in the groovysh, then leave off the "def"

Groovy Execute

- Commandline executions are easily called from Groovy
- Can be a useful hack in a build script

```
// unix / mac
println 'ifconfig'.execute().text

// windows
println 'ipconfig'.execute().text
```

Groovy Collection Operations

```
[1, [2,3]].flatten() // [1, 2, 3]
['a', 'b'].each { item -> println item }
['a', 'b'].collect { it + '1' } // ['a1', 'b1']
['a', 'b', 'c'].findAll { it != 'c' } // ['a', 'b']
[1, 2, 3].every { it < 3 } // false
[1, 2, 3].any { it < 3 } // true
// many more
```

- Gradle's API can stay light as we don't need provide as much convenience methods as with Java
- Learning Groovy has many benefits. It is a powerful tool for many purpose (e.g. testing).
- The book Groovy in Action is the standard reference for Groovy.

Hashmaps

```
def food = [:]

food.vegetables = ["peas", "green beans"]
food.fruit = ["apples", "oranges", "kiwi"]
```

- Hashmap keys are dynamically added to the map

String Interpolation

```
def food = [:]

food.vegetables = ["peas", "green beans"]
food.fruit = ["apples", "oranges", "kiwi"]

println food

println "Today: ${new Date()}"
print "fruits: ${food.fruit} "
println "and veggies: ${food.vegetables}"
```

- Referred to as “GStrings”
  - Groovy Strings

Spread Operator

```groovy
def languages = []

languages << "Java"
languages << "Groovy"

println languages
println languages*.toUpperCase()
```

- Invokes method on each element of the collection
- Great in Gradle when working with Configurations

Gradle Build Scripts

- Must be compilable by Groovy.
- Can’t be executed by plain Groovy runtime.
- Delegate to an associated org.gradle.api.Project object.

```
// does not compile
println 'Gradle

// compiles, fails when run with plain Groovy
println name

// compiles, fails when run with Groovy or Gradle
println zipCode
```

Method Pointers and DSL

- Groovy provides a way to have a reference to an objects method
- Creates a great way to create a DSL

```
def shoppingList = []
def add = shoppingList.&add
def remove = shoppingList.&remove

add "Milk"
add "Bread"
add "Beer"
remove "Beer"
add "Apples"

print shoppingList
```

Gradle Build Scripts

- **Configure** the Project object.
- Do **not** execute the build.

Labs

- Lab 05 - Groovy

Java Plugin

Plugins

- Plugins == Build Scripts
- Two Flavors:
  - Another build script (local or remote) (Script Plugin)
  - A class implementing org.gradle.api.Plugin (Binary Plugin)

Applying Plugins

- Any gradle script can be a plugin.
- Binary plugins must be in the build script classpath
  - can have id's (meta properties in the jar).
  - will learn later how to add elements to the build script classpath.
  - The build-in plugins are by default in the build script classpath.

```
apply from: 'otherScript.gradle'
apply from: 'http://mycomp.com/otherScript.gradle'
```

```
apply plugin: org.gradle.api.plugins.JavaPlugin
apply plugin: 'java'
```

What Plugins Can Do

- Configure the project object (e.g. add task instances)
- Add other classes to classpath (e.g. custom task types)
- Add props and methods to the project object (extend DSL).
- Build Script Decomposition
  - Separate Imperative from Declarative
  - Modularization
- Code Reuse

Standard Gradle Plugins

Plugin-Id	applies
`base`	
`java-base`	`base`
`java`	`java-base`
`groovy-base`	`java-base`
`groovy`	`groovy-base`
`scala-base`	`java-base`
`scala`	`scala-base`
`war`	`java`
`osgi`	
`code-quality`	
`maven`	
`eclipse`	

Applying the Java Plugin

- With NO dependencies
- Following Maven project structures

```
apply plugin: 'java'
```

```
apply plugin: org.gradle.api.plugins.JavaPlugin
```

Clean Task

- By default clean deletes the buildDir
- You can specify additonal files to delete

Name	Type
`clean`	`Delete`

```
clean {
  delete 'fooDir', 'bar.txt',
     fileTree('texts').matching { ... }
}
```

Compile Tasks

- Usually configured via the source set.
- Provides all the options of the Ant javac task

Name	`compile, testCompile`
Type	`Compile`
Input	`sourceSets.main(test).java` `configurations.compile(testCompile)`

```
compileJava {
  options.fork {
    memoryMaximumSize = '512M'
  }
}
```

Test Task

- Support for JUnit and TestNG
- Parallel Testing
- Custom Fork Frequency
- Remote Listeners
- Tests auto-detected in `sourceSets.test.classes`

Name	`test`
Type	`Test`
Input	`sourceSets.test.classes` `configurations.testRuntime`

Test Task Example

```
test {
   jvmArgs ["-Xmx512M"]
   include "**/tests/special/**/*Test.class"
   exclude "**/Old*Test.class"
   forkEvery = 30
   maxParallelForks = guessMaxForks()
}

def guessMaxForks() {
     int processors =
        Runtime.runtime.availableProcessors()
     return Math.max(2, (int) (processors / 2))
}
```

Disables Auto Detection

Test Task Listeners

```
test {
  beforeTest { descr ->
    // do something
  }
  afterTest { descr, result ->
    // do something
  }
  afterSuite { descr, result ->
    // do something
  }
}
```

Local Dependencies

Ant Style

```
apply plugin: 'java'

dependencies {
  compile fileTree(dir: 'lib',  includes: ['*.jar'])
}

sourceSets.main.java.srcDir = 'src'
```

Running a Single Test

```
gradle -Dtest.single=ThisUniquelyNamedTest test

or

gradle -Dtest.single=*IntegrationTest test
```

Labs

- Lab 06 - Applying Plugins
- Lab 07 - Tests

Ant

Ant

- Ant is Gradle's friend not its competitor.
- Gradle uses Ant task's internally.
- You can use any Ant task from Gradle.
- Ant tasks are an integral part of Gradle.
- Gradle ships with Ant.
- You can import any Ant build into Gradle

Ant Tasks

- Gradle provides an instance of the Groovy AntBuilder

```
ant.delete dir: 'someDir'
ant {
  ftp(server: "ftp.comp.org", userid: 'me', ...) {
    fileset(dir: "htdocs/manual") {
      include name: "**/*.html"
    }
    // high end
    myFileTree.addToAntBuilder(ant, 'fileset')
  }
  mkdir dir: 'someDir'
}
```

Importing Ant Builds

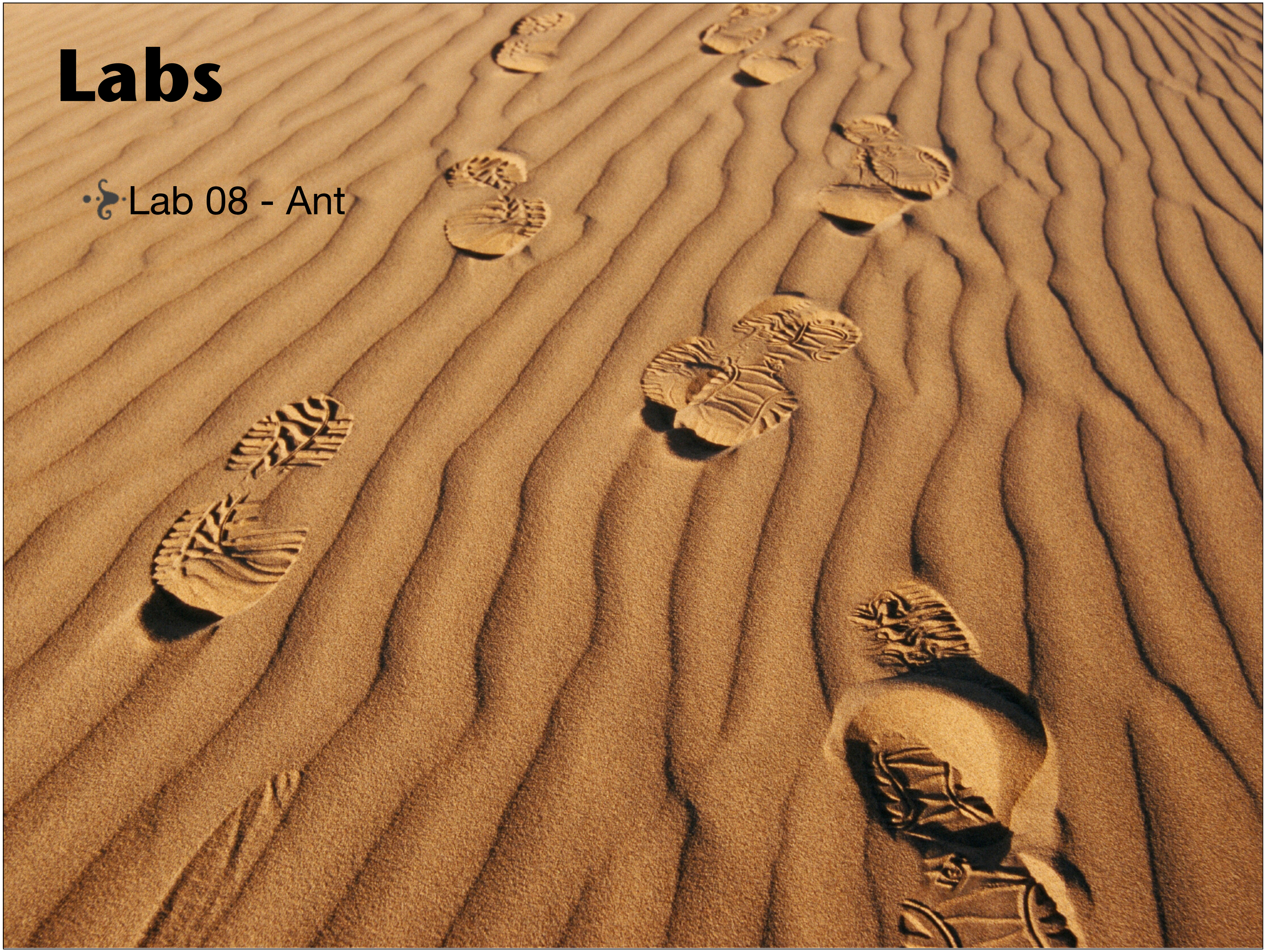
```xml
<project>
  <target name="hello" depends="intro">
    <echo>Hello, from Ant</echo>
  </target>
</project>
```

```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
```

Labs

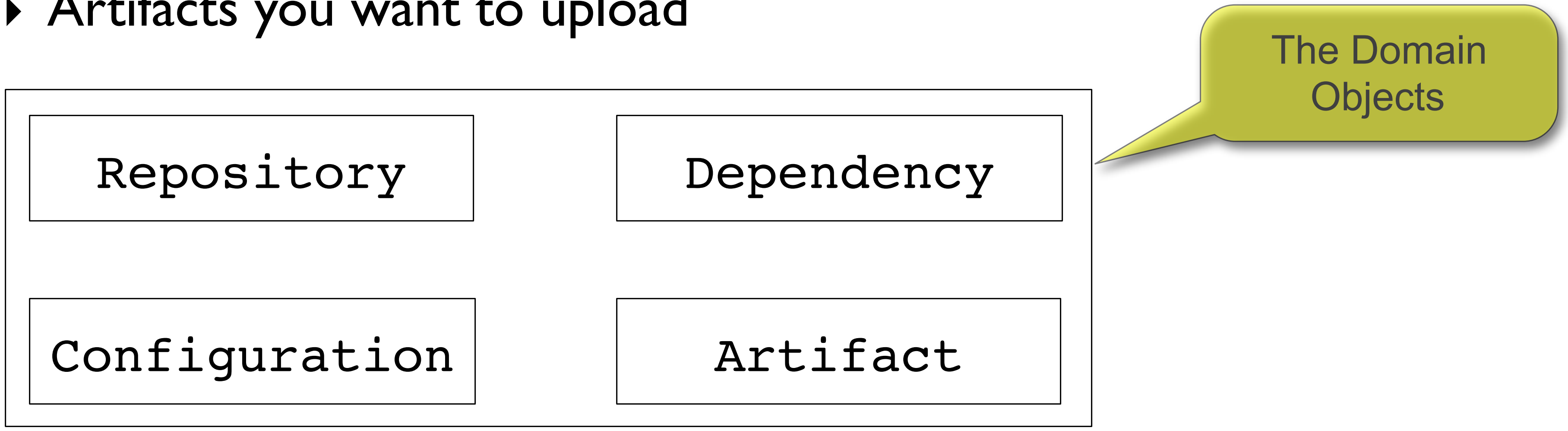
- Lab 08 - Ant



Dependencies

Dependencies

- Repository dependencies
  - e.g. from mavenCentral
  - with module descriptors (pom.xml/ivy.xml)
- Repository-less dependencies (specified by path).
- Projects dependencies in a multi-project build.
- Artifacts you want to upload



Dependencies

```
apply plugin: 'java'
repositories {
  mavenCentral()
}
dependencies {
  compile "junit:junit:4.4"
  compile group: 'junit', name: 'junit',
      version: '4.4'
  compile files('file1.jar'), fileTree('lib'),
         project(':otherProject')
}
```

String/Map ~ Repository Dependency

FileCollection/Tree ~ Repository Less Dependency

Project ~ Project Dependency

Dependencies & Java Plugin

```
apply plugin: 'java'
configurations { myConf.extendsFrom compile }
dependencies {
  compile "junit:junit:4.4"
  runtime org:'asm', name:'asm-all', version:'3.2'
  testCompile files('file1.jar')
  myConf "log4j:log4j:1.2.9"
}
```

- The Java plugin adds configurations.
- Many Java plugin tasks use those configurations as default input values (e.g. test).
- Configurations can extend each other

Transitive Dependencies

- Exists for repository dependencies.
- pom.xml/ivy.xml does describe transitive dependencies.
- Default version conflict resolution is newest.
- Transitive resolution is customizable.

```
dependencies {
  compile "org.hibernate:hibernate:3.1" {
    force = true
    exclude module: 'cglib'
  }
  compile "org:somename:1.0" {
    transitive = false
  }
}
configurations.myconf.transitive = false
```

Repositories

- Any Maven/Ivy repository can be accessed.
- Very flexible layouts are possible for non Maven repositories.

```
repositories {
  mavenCentral()
  mavenCentral(urls: ['http://repo.com'])
  mavenRepo(urls: ['http://repo1.com',
           "http://repo2.com"])
  flatDir(dirs: [“dir1”, “dir2”])
}
```

Labs

- Lab 09 - Dependencies
- Lab 10 - War Project

Beyond Gradle Basics

Deep API

- Gradle let you customize its domain objects:
  - Enhance their API
  - Define rules for how they should be constructed

Global Properties

```
myDocsDestDir = "$buildDir/myDocs"

task myDocs << {
  copy {
    from 'someDir'
    into myDocsDestDir
  }
}

task zip(type: Zip) {
  from myDocsDestDir
}
```

Dynamic Properties

```
task myDocs {
  destDir = "$buildDir/myDocs"
  doFirst {
    copy {
      from 'someDir'
      into destDir
} } }
task zip(type: Zip) {
  from myDocs.destDir
}
```

Adds a dynamic property

- Applicable to most Gradle types
- Good OO design (e.g. encapsulation)
- Custom task is an alternative (more heavyweight)

Dynamic Methods

```
task bar {
  serviceUrl = ...
  domainGroup = {
     getGroup(serviceUrl)
  }
}
task foo {
  fooProp = bar.domainGroup()
}
```

Adds a dynamic method

- Providing methods via a closure property is a Groovy trick.
- You can also mix-in any Java/Groovy object
  - Beyond the scope of this class.

Domain Object Container

- Handler for most domain objects (plugins, deps, tasks, ...)

Contained in allJars

Build-In Filter

```
allJars = tasks.withType(Jar)
task myJar(type: Jar)

webTasks = tasks.matching {
   task->task.name.startsWith('web')}

compJars = tasks.withType(Jar).matching { task ->
   task.name.startsWith('comp')}

task buildAllJars(dependsOn: allJars)
```

Custom Filter

Filter Chaining

Dynamic Depends

Configuration Rules

- Provided by the domain object container

```
tasks.allObjects { task ->
  task.doFirst { println 'rule for all tasks' }
}
tasks.withType(Jar).allObjects { jar ->
  jar.destinationDir = 'somePath'
  jar.doLast { /* do something */ }
}

tasks.whenAdded { task -> ... }
```

Init Scripts

- Init scripts are run before the build starts:
- Set up properties based on the current environment
- Define machine specific details, such as where JDKs are installed.
- Register build listeners.
- Enhance builds you don't want to touch.
- GRADLE_USER_HOME/init.gradle is automatically applied as an init script.
- You can specify any init script via the -I command line option.

```
>gradle assemble -I ci-init.gradle
```

Sample Init Script

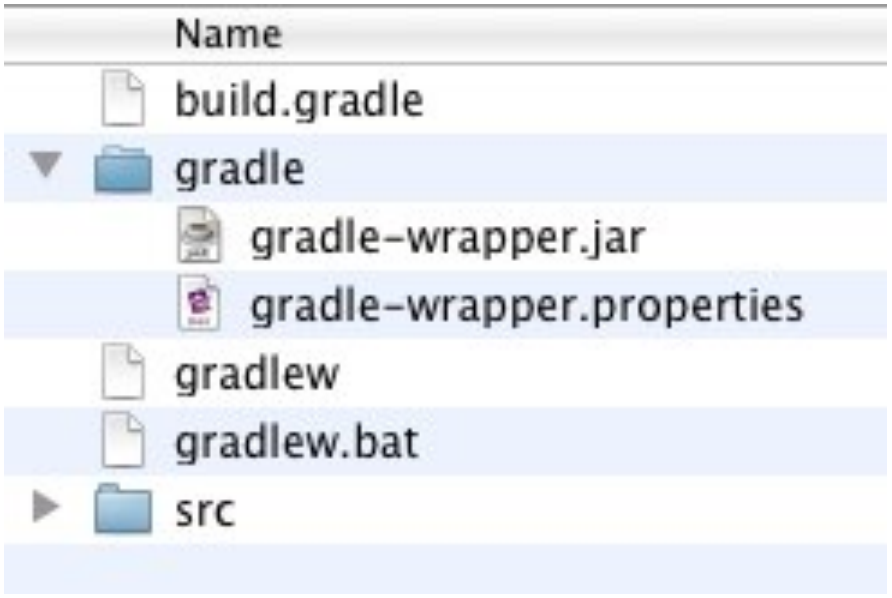
```
initscript {
  repositories {
    mavenCentral()
  }
  dependencies {
    classpath 'org.apache.commons:commons-math:2.0'
  }
}
gradle.startParamter // do something with them
gradle.addBuildListener ...
```

Wrapper Task

- Wrapper task generates:
  - wrapper scripts
  - wrapper jar
  - wrapper properties.

```
task wrapper(type: Wrapper) {
  gradleVersion = '0.6'
  jarPath = 'gradle'
}
```

Wrapper Files


Name
build.gradle
gradle
gradle-wrapper.jar
gradle-wrapper.properties
gradlew
gradlew.bat
src

```
>./gradlew assemble
```

Multi-project Builds

Multi-Project Builds

- Arbitrary Multiproject Layout
- Configuration Injection
- Project Dependencies & Partial builds
- Separate Config/Execution Hierarchy

Configuration Injection

- **ultimateApp**
  - api
  - webservice
  - shared

```
subprojects {
    apply plugin: 'java'
    dependencies {
        compile "commons-lang:commons-lang:3.1"
        testCompile "junit:junit:4.4"
    }
    test {
        jvmArgs: ['Xmx512M']
    }
}
```

Filtered Injection

- **ultimateApp**
  - api
  - webservice
  - shared

```
configure(nonWebProjects()) {
  jar.manifest.attributes
     Implementor: 'Gradle-Inc'
}

def nonWebProjects() {
  subprojects.findAll {project ->
    !project.name.startsWith('web')
  }
}
```

Project Dependencies

- ultimateApp
  - **api**
  - webservice
  - shared

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

First Class Citizen

Partial Builds

- ultimateApp
  - **api**
  - webservice
  - shared

```
>gradle build
>gradle buildDependents
>gradle buildNeeded
```

There is
no one-size-fits-all
project structure
for the
enterprise.

The physical structure of your projects should be determined by **your requirements**.

Name Matching Execution

- **ultimateApp**
  - api
  - webservice
  - shared

```
>gradle build
>gradle classes
>gradle war
```

Task/Project Paths

- For projects and tasks there is a fully qualified path notation:
  - `:` (root project)
  - `:clean` (the clean task of the root project)
  - `:api` (the api project)
  - `:services:webservice` (the webservice project)
  - `:services:webservice:clean` (the clean task of webservice)

```
>gradle :api:classes
```

Defining a Multi Project Build

- `settings.gradle` (location defines root).
- root project is implicitly included

Defines a virtual hierarchy

By default maps to file path `<root>/project1`

```
include 'project1','project2','project2:child1'

// Everything is configurable
rootProject.name = 'main'
project(':project1').projectDir = '/myLocation'
project(':project1').buildFileName =
  'project1.gradle'
```

Default to root dir name

Default to build.gradle

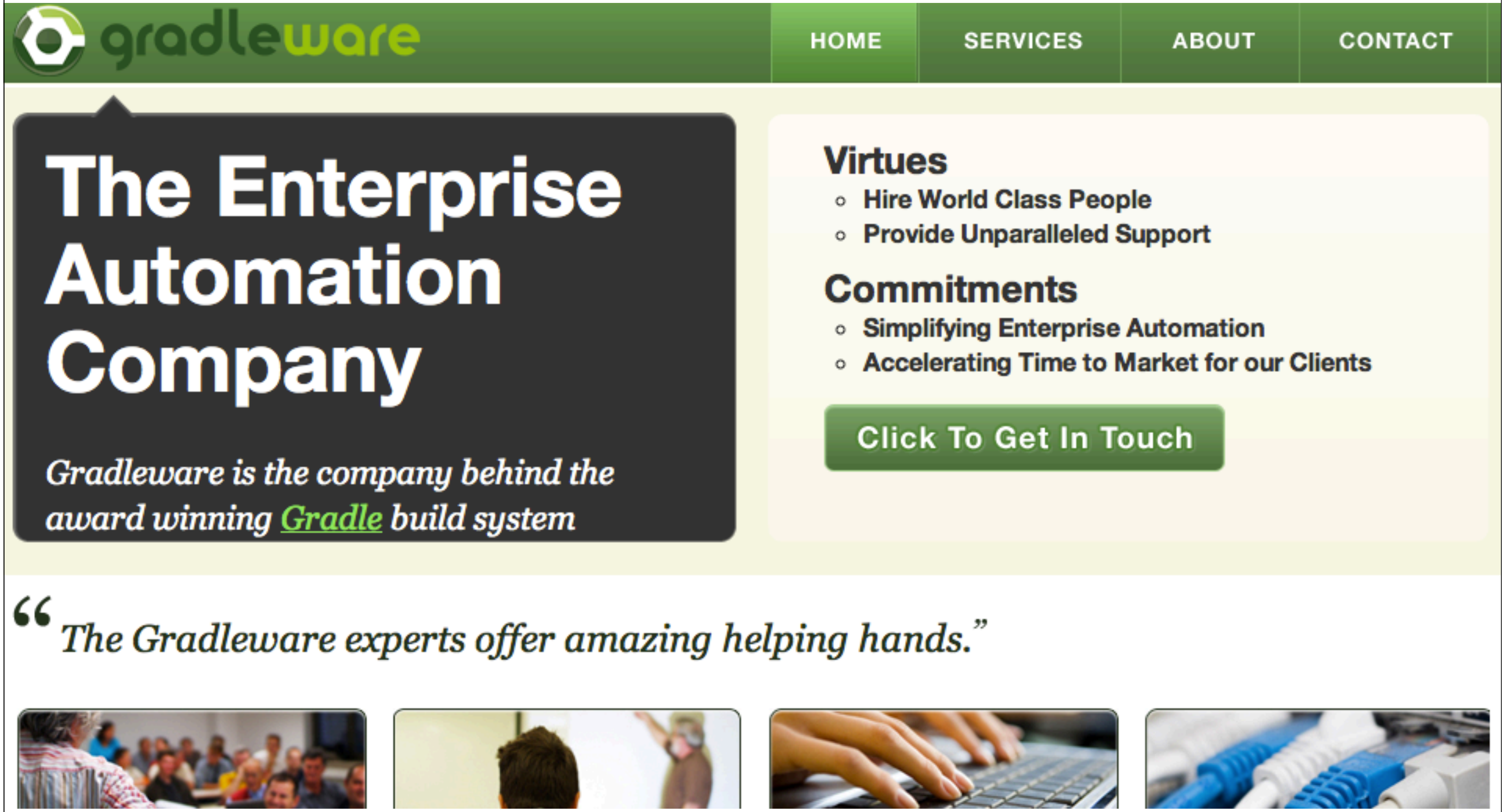
Labs

- Lab 11 - Multi-Project Build

Didn’t Talk About it

- Smart Merging
- Smart Exclusion
- Skipping Tasks
- Conditional Tasks
- Hooks
- Ivy
- Custom Tasks

Commercial Support: *gradleware.com*



The Enterprise Automation Company

Gradleware is the company behind the award winning Gradle build system

Virtues

- Hire World Class People
- Provide Unparalleled Support

Commitments

- Simplifying Enterprise Automation
- Accelerating Time to Market for our Clients

"The Gradleware experts offer amazing helping hands."