Welcome to the Gradle Workshop - Setup

- For some features in the workshop, you will need at least Gradle 2.14.
- Download gradle from
  - https://services.gradle.org/distributions
- You will also need a recent version of Git and an SDK (1.8 or later is preferred)
- You can download Git for your platform from
  - http://git-scm.org
- If you want to try the optional lab for integration with Jenkins, you will need a Jenkins instance up and running with the Gradle plugin installed
  - http://jenkins-ci.org
- http://github.com/brentlaster/docs

Learn to Use Gradle (and understand it) Workshop

Brent Laster

RWX, 2016

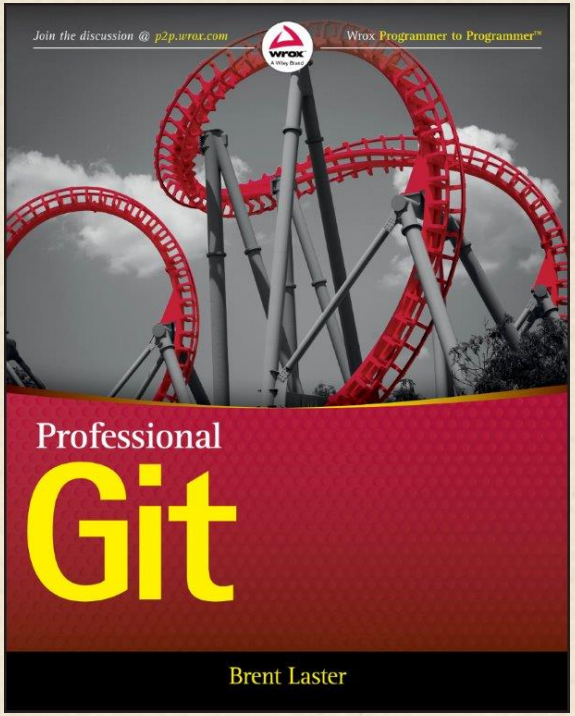
Welcome!

About me

- Senior Manager, R&D at SAS in Cary, NC
- Part-time trainer
- Git, Gerrit, Gradle, Jenkins
- Author - NFJS magazine, Professional Git book
  Brent.Laster@sas.com
- LinkedIn: https://www.linkedin.com/in/brentlaster
- Twitter: @BrentCLaster

Professional Git - available 12/12/16

- **Available on:**
  - Amazon.com
    - » Check price
  - Wiley.com
    - » (discount for Rich Web Experience attendees)
      - » 35% off
      - » Use code: GIT35



Agenda

- About Gradle
- Tasks
- Build Scripts
- Build Lifecycle and Phases
- Properties and Plugins
- Wrapper and Daemon
- Dependency Management
- Testing
- SourceSets
- Continuous Builds
- Multi-project Builds
- Using Gradle with Jenkins

Why Gradle?

- General Purpose Build *Automation* System
- Middle ground between ANT and Maven
  - ANT – maximum flexibility, no conventions, no dependency management (offset by Ivy)
  - Maven – strong standards, good dependency management, hard to customize, not flexible
- Offers useful “built-in” conventions, but also offers ease of extension
- Useful for developing build standards
- Rich, descriptive language (DSL – Groovy) – Java core
- Kotlin support in the works
- Built-in support for Groovy, Java, Scala, OSGI, Web

Why Gradle?

- Multiple dependency management models
  - Transitive – can declare only the top-level dependencies and interface with Maven or Ivy
  - Manual – can manage by hand
- Toolkit for domain-specific build standards
- Offers useful features – upstream builds, builds that ignore non-significant changes, etc.

About Gradle

- License – Apache v2
- Active development and community support
- Dedicated team of developers – Hans Dockter, Adam Murdoch
- Considers itself as a build automation framework
- Commercial support through Gradleware company

Installing Gradle

- Requires JDK 1.5 or higher
- Ships with its own Groovy library
- Download from http://www.gradle.org/downloads.html
- add *GRADLE_HOME*/bin to your PATH environment variable
- To check if Gradle is properly installed just type **gradle –v**
- VM options for running Gradle can be set via environment variables.
  - You can use GRADLE_OPTS or JAVA_OPTS. Those variables can be used together.
  - A typical use case would be to set the HTTP proxy in JAVA_OPTS and the memory options in GRADLE_OPTS.
  - Those variables can also be set at the beginning of the **gradle** or **gradlew** script.

Everything's Groovy

- What is Groovy?
  - OO language for the Java platform
  - Like Java + easier to use, more dynamic features
  - Features similar to some found in Python, Ruby, Perl, Smalltalk
  - Can be used as a scripting language
  - Compiles down to JVM bytecodes so interoperates with Java
- Why Groovy?
  - Why not XML
    - » XML is good for nested relationships, but not good for flow or large efforts
  - Groovy is similar to Java
  - Every Gradle build file is an executable Groovy script
  - Use of Groovy allows you to do general purpose programming tasks in the build file
    - » Helps alleviate special case handling in ANT or plugin development for Maven to handle unusual cases
- Groovy Setup
  - Download and unzip groovy-*.zip to dir
  - Add GROOVY_HOME env var set to dir
  - Add $GROOVY_HOME/bin to PATH

Groovy is Java and Gradle is Groovy

Java

```
import java.io.File;

String parentDir = new File ("abc.txt")
                        .getAbsoluteFile()
                        .getParentPath()
```

Groovy

```
def parentDir = new File("abc.txt").absoluteFile.parentPath
```

Gradle

```
parentDir = file("abc.txt").absoluteFile.parentPath
```

Groovy, Gradle, and Domain Specific Languages (DSLs)

- Remember, every Gradle build file is an executable Groovy script
- Pros
  - You can easily code up things for simple cases (iterators) or more complex cases
  - Can drop into Groovy code wherever needed
- Cons
  - Need to know Groovy
  - As with any special-casing, can end up being a maintenance support issue
- Happy medium
  - Gradle provides a DSL targeted for build efforts
  - No requirement to know Groovy
  - Can extend DSL through plugins
    - » Add new tasks definitions,
    - » Change behavior of existing tasks
    - » Add new objects
    - » Create new keywords

Closures

- **A block of Groovy code inside {}.**
- **From docs.groovy-lang.org**

> "A closure in Groovy is an open, anonymous, block of code that can take arguments, return a value and be assigned to a variable. A closure may reference variables declared in its surrounding scope. In opposition to the formal definition of a closure, Closure in the Groovy language can also contain free variables which are defined outside of its surrounding scope."

- **Can be passed around just like any other object.** Passing closures to methods is a common Groovy idiom.
  - » Instance of groovy.lang.closure
- **The last statement of a closure is the closure's return value, even if no return statement is given.**
- A Groovy method containing a single expression is a function that returns the value of that expression.
- More info: http://docs.groovy-lang.org/latest/html/documentation/index.html#_closures

Gradle/Maven Standard Layout

- src/main/java – java classes
- src/main/groovy – groovy classes
- src/main/webapp - web files
- src/test/java | src/test/groovy - unit tests
- plugins for others

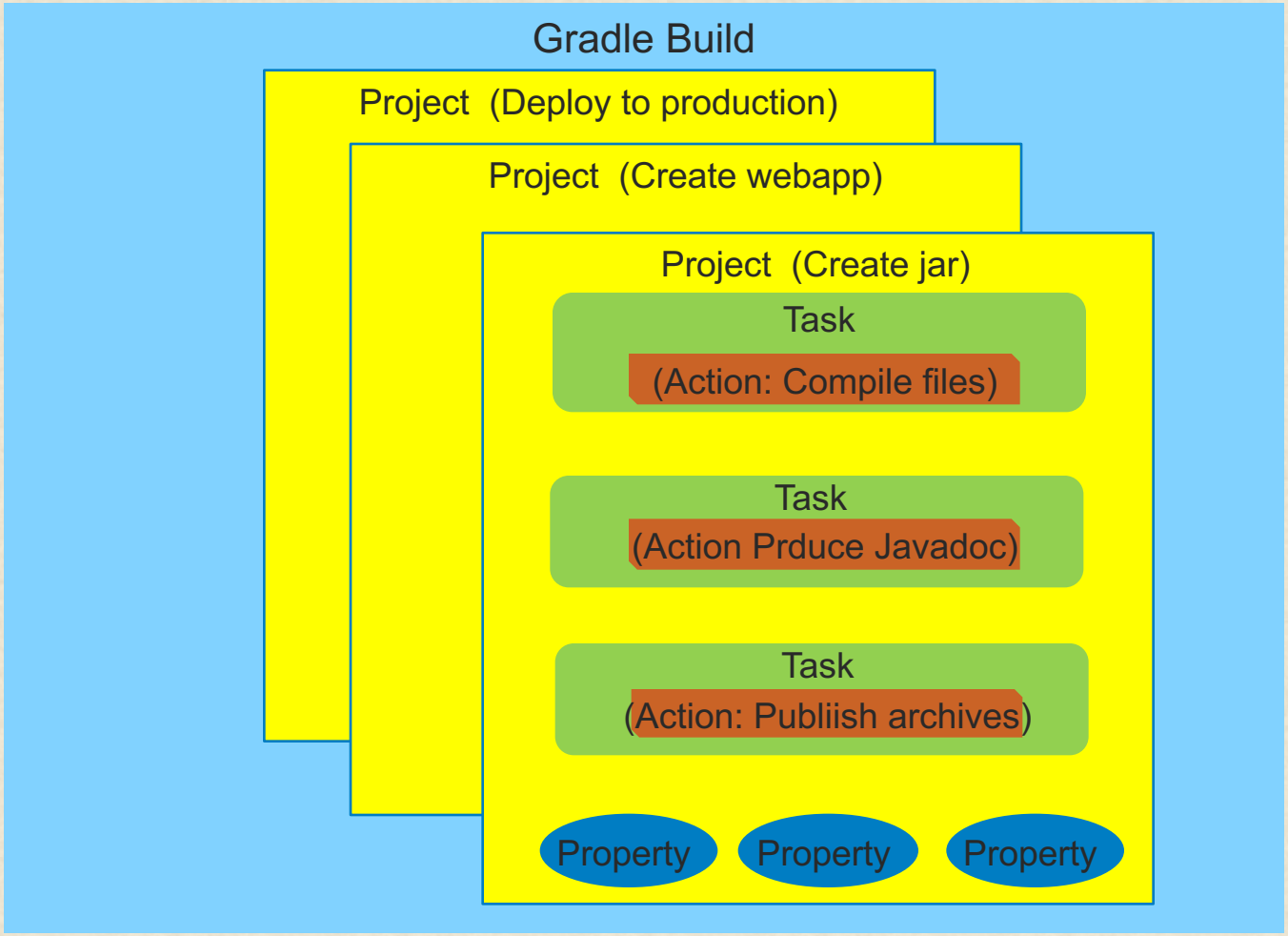
Gradle Command Line

- invoke Gradle via gradle [option(s)] [task(s)]
- Useful options
  - -h --help – show basic help info
  - -b –build-file – specify a non-default build file (not build.gradle)
  - -d –debug – log in debug mode
  - --daemon – use the gradle daemon to run builds – start it if needed
  - -i --info – set log level to info
  - -q –quiet – log errors only
  - --profile – profiles build execution time and generates a report
- Other
  - properties - Emits all the properties of the build's Project object. The Project object is an object representing the structure and state of the current build.
  - tasks - Emits a list of all tasks available in the current build file. Note that plug-ins may introduce tasks of their own, so this list may be longer than the tasks you have defined.

Gradle Build Script Basics

- Two basic concepts – projects and tasks
- Project
  - Every Gradle build is made up of one or more projects
  - A project is some component of software that can be built (examples: library JAR, web app, zip assembled from jars of other projects)
  - May represent something to be done, rather than something to be built. (examples: deploying app to staging area or production)
- Task
  - Represents some atomic piece of work which a build performs (examples: compiling classes, creating a jar, generating javadoc, publishing archives to a repository)
  - List of actions to be performed
  - May include configuration

Gradle Structure



Anatomy of a Task

```
def destDirs = new File('target3/results')
```

KEYWORD TASK NAME TYPE SPECIFICATION

```
task createTarget(type: DefaultTask) {
```

API

```
    doLast {
        destDirs.mkdirs()
    }
}
```

ACTION

SHORTHAND FOR DefaultTask AND doLast

```
task createTarget << {
    destDirs.mkdirs()
}
```

Tasks as Objects

- In Gradle, tasks are objects
- Tasks have a type
  - Default type is DefaultTask
  - All tasks descend from DefaultTask object type
  - May be derived from custom task types
  - All tasks implement the Task interface
  - Lots of built-in Task types
- Tasks have an API (methods and properties)
  - Callable outside of task definition

```
task grantAccess <<  { println ‘Access Granted!’ }

grantAccess.onlyIf { System.properties['user.name'] == ‘admin' }
```

Built-in Task Types

- **Copy**
  - Copies files from source to destination
  - File patterns can be added to include or exclude files
  - Creates destination directory if doesn't exist
- **Java**
  - Creates a jar from source files
  - Java plug-in creates a task of this type (named "jar")
  - Packages source and resources together with basic manifest
  - Jar has project name
  - Placed in build/libs directory
  - Easily configure archive name and destination directory
  - Manifest can be populated with custom attributes via readable Groovy map syntax
  - Contents of the JAR are identified by the from sourceSets.main.classes line, which specifies that the compiled .class files of the main Java sources are to be included.
- **War**
  - Copies contents of src/main/webapp into WEB-INF/classes
  - Copies runtime dependencies into WEB-INF/lib
  - Only groovy-all jar is included
    - » Since others are indicated as providedCompile
    - » Including Groovy as a dependency, resulting web application can deployed in any normal Java environment

Built-in Task Types

- **JavaExec**
  - Runs a Java class with a main() method
  - Allows you to integrate command line java invocation into builds
- **Delete**
  - Deletes files or directories
  - Properties:
    - » delete – The set of files which will be deleted by this task
    - » targetFiles – The resolved set of files which will be deleted by this task
  - Methods:
    - » delete – Add files to be deleted by this task
  - Example:

```
task cleanTemp(type: Delete) {
        delete ‘tempFolder', ‘tempFile‘
}
```

Creating Tasks Dynamically

- Can use parameter substitution (Groovy syntax)

```
4.times { counter ->
        task "task$counter" << {
                println “This is task number $counter“
        }
}
```

- This defines 4 tasks you can invoke: task0, task1, task2, and task3
- Invoke the usual way

```
gradle task#
```

- Equivalent to writing this in build script:

```
task task0 << { println “This is task number 0”}
task task1 << { println “This is task number 1”}
task task2 << { println “This is task number 2”}
task task3 << { println “This is task number 3”}
```

Gradle's Task API (Methods and Properties)

- Methods available in DefaultTask type include:
  - » dependsOn(task) – sets task to be a dependency (called before) Example: task foo { dependsOn bar1, bar2 }
  - » doFirst(closure) – Adds a block of code to the beginning of a task
  - » doLast(closure) – Adds a block of code to the end of a task
  - » onlyIf(closure) – Allows you to express a predicate to determine if the task should be executed
- Properties available in DefaultTask type include:
  - » didWork – boolean indicates task completion status
  - » enabled – boolean to set for whether to run task or not
  - » path – string that indicates fully qualified path of a task
  - » logger – reference to Gradle's logger
  - » logging – reference to Gradle's logging level
  - » description – string of human-readable text
  - » temporaryDir – file object pointing to a temporary directory

Example "Hello World" Gradle Build Script

- Named build.gradle

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

- Invoke via gradle –q hello
  - Script defines a single task called hello
  - Defines action associated with it
  - On invocation, runs task and then action associated with it
  - "task" as differentiated from Ant's "target"

Lab 1 – Creating and invoking a simple Gradle Build Script

UP-TO-DATE?

- What does gradle mean when it says a task is UP-TO-DATE?
  - Gradle is an incremental build system - it checks to see if a task really needs to be run before running it.
  - Gradle determines if a task actually needs to be executed by checking its (the task's) inputs and outputs.
  - Saves time, particularly on large builds.
- Example
  - Task to do a compile: if no source files have changed and output hasn't been deleted, then is considered UP-TO-DATE.
- Bypass
  - Change input or output
  - Use command line option --rerun-tasks

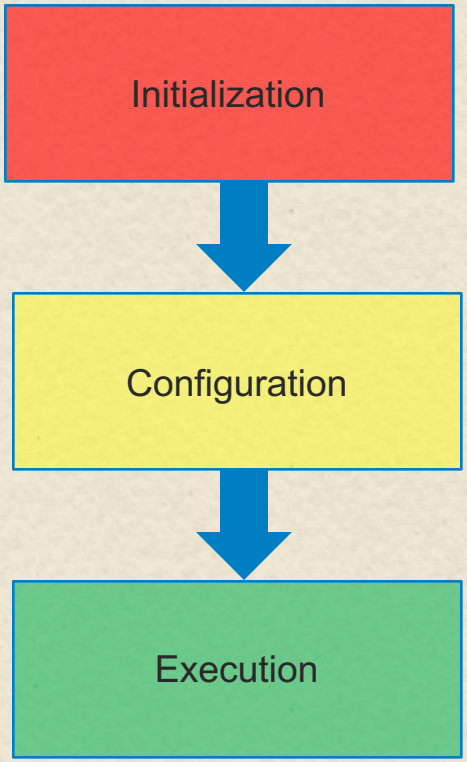
26

The Gradle Build Lifecycle

- “the core of Gradle is a language for dependency based programming”
- User defines tasks and dependencies between tasks
- Gradle guarantees:
  - Tasks are executed in the order of their dependencies
  - Each task is executed only once
- A Gradle build has 3 distinct phases
  - Initialization
  - Configuration
  - Execution

Gradle Build Phases

- Initialization
  - During the initialization phase, Gradle determines which projects are going to take part in the build, and creates a Project instance for each of these projects
- Configuration
  - During this phase the project objects are configured.
  - Gradle has an incubating opt-in feature called 'configuration on demand'. In this mode, Gradle configures only relevant projects.
    - » Can save time in large multi-project builds.
    - » Enabled in gradle.properties
- Execution
  - Gradle determines the subset of the tasks, created and configured during the configuration phase, to be executed.
  - Subset is determined by the task name arguments passed to the **gradle** command and the current directory.
  - Gradle then executes each of the selected tasks.



28

Configuration vs. Execution

- Think of configuration as applying to anything in the build script that doesn't call a task's API.
- Think of execution as applying to anything in the build script that does call a task's API (creates an "action").

Example build.gradle

```
println  '1. In configuration phase - glob
task abc {
                  println '2. In configuratic
}

abc.doFirst {
                 println '3. In execution ph
}

task xyz {
                 println "4. In configuration
                doLast { println "5. In exe
}
```

Not in <task>.<api> call

Not in <task>.<api> call

In <task>.<api> call (doFirst)

Not in <task>.<api> call

In <task>.<api> call (doLast)

```
> gradle
1. In configuration phase - global
2. In configuration phase - abc
4. In configuration phase - xyz
```

```
> gradle abc
1. In configuration phase - global
2. In configuration phase - abc
4. In configuration phase – xyz
:abc
3. In execution phase - abc
```

```
> gradle xyz abc
1. In configuration phase - global
2. In configuration phase - abc
4. In configuration phase – xyz
:xyz
5. In execution phase - xyz
:abc
3. In execution phase - abc
```

Adding Calls to a Task

```
def destDirs = new File('target3/results')
task createTarget(type: DefaultTask) {
    doLast {
            destDirs.mkdirs()
            }
}
```

- Once a task is defined, you can add separate API calls outside of the main closure.

```
createTarget.doFirst {
            destDirs.mkdirs()
}
createTarget.doLast {
            destDirs.mkdirs()
}
createTarget << {
            destDirs.mkdirs()
}
```


30

Lab 2 – Understanding Gradle Phases

The Properties of Properties

- Along with projects and tasks, properties are one of the main components of a Gradle program/script.
- Properties are not the same as local variables.

Local Variables

Feature of Groovy, not Gradle

MINGW32:/c/Users/sasbcl/playpen

```
$ cat local.gradle

def source = "myinput"
def target = "myoutput"

task copySource (type: Copy) {

        from source
        into dest

}
```

- Gradle comes with standard properties
- Can define “extra” properties

Standard Properties in Gradle

- Provided by Project object

Name	Type	Default Value
project	Project	The Project instance
name	String	The name of the project directory.
path	String	The absolute path of the project.
description	String	A description for the project.
projectDir	File	The directory containing the build script.
buildDir	File	*projectDir*/build
group	Object	unspecified
version	Object	unspecified
ant	AntBuilder	An AntBuilder instance

standard_props.gradle - Notepad

File Edit Format View Help

```
description = "Print some standard properties"

println "Build directory: $buildDir"
println "Description: $project.description"
```

MINGW32:/c/Users/sasbcl/playpen

```
sasbcl@L7B871 ~/playpen
$ gradle -q -b standard_props.gradle
Build directory: C:\Users\sasbcl\playpen\build
Description: Print some standard properties

Welcome to Gradle 2.3.

To run a build, run gradle <task> ...

To see a list of available tasks, run gradle tasks

To see a list of command-line options, run gradle --he

To see more detail about a task, run gradle help --tas

sasbcl@L7B871 ~/playpen
$
```

Non-standard Properties

- What happens if we spell it wrong?
- In the olden days…
  - It would have dynamically created a property named “descript”
- Today…
  - Build failure: No such property
- What if we want to create a new custom property?
  - Suppose we create a “release_note” property…
- Gradle provides the “ext” (extra) namespace to define custom variables
  - Provides validation - checking properties for defined names

standard_props.gradle - Notepad

File Edit Format View Help

```
ext.release_note = "Coolest release ever!"

println "Release note =  $release_note"
```

MINGW32:/c/Users/sasbcl/playpen

```
To see more detail about a task, run gradle help --tas

sasbcl@L7B871 ~/playpen
$ gradle -q -b standard_props.gradle
Release note =  Coolest release ever!

Welcome to Gradle 2.3.

To run a build, run gradle <task> ...

To see a list of available tasks, run gradle tasks

To see a list of command-line options, run gradle --he

To see more detail about a task, run gradle help --tas

sasbcl@L7B871 ~/playpen
$
```

Custom Properties in Gradle

- Define these “extra” properties in ext{ } script block in build file or prefix with ext.

```
ext.uniqueProperty = ‘only 1’ or
ext {
        uniqueProperty= ‘only 1’
    }
```

custom_prop2.gradle - Notepad

```
ext {
      release_note = "Coolest release ever!"
    }

println "Release note = $release_note"
println "Release note = $ext.release_note"
println project.release_note
```

- Property is automatically added to internal project properties

gradle.properties - Notepad

```
release_note='Best release ever!'
```

- Multiple ways to access property in script

```
println uniqueProperty
println ext.uniqueProperty
println project.uniqueProperty
```

custom_prop.gradle - Notepad

```
println "Release note =  $release_note"
```

- If not defined in build file
  - Can be passed in via command line -P option
  - Or put in gradle.properties file.
  - In both cases above, custom property is automatically added to ext namespace

MINGW32:/c/Users/sasbcl/playpen

```
sasbcl@L7B871 ~/playpen
$ gradle -q -b custom_prop.gradle
Release note =  'Best release ever!'

Welcome to Gradle 2.3.
```

Setting Properties outside of a Build Script

- Multiple methods
  - -D
    - pass a system property to the JVM that runs Gradle
    - Format: -Dorg.gradle.project.property.Name=value
  - Passing properties to a project
    - external properties file – put gradle.properties file in home directory or in project directory
      - home dir default = USER_HOME/.gradle
      - Format of line: property = value
    - -P
      - add an extra property to a build
      - set a value for an existing property
      - Format is –PpropertyName=value
    - Environment variable
      - Useful when we have environment restrictions and to avoid lots of retyping
      - Format: ORG_GRADLE_PROJECT_propertyName
      - Example
        - export ORG_GRADLE_PROJECT_fileVersion=3.2

          in script

          println “File Version: $fileVersion”
  - another build script
    - apply from: “script name”, to: <arbitrary object>
- Gotcha – if plugin is applied, properties must be set after apply statement

Passing Values to Properties During Invocation

- Define property and value on gradle invocation

  gradle –Dmy.property=value

- Reference value via System.properties[my.property]

```
task helloWorld << { println 'Hello World!' }
helloWorld.doFirst { println 'User name = ' + System.properties['user.name'] }
helloWorld.onlyIf { System.properties['user.name'] == 'Bob' }

C:\gradle2>gradle -Duser.name=Bob helloWorld
:helloWorld
User name = Bob
Hello World!
BUILD SUCCESSFUL
C:\gradle2>gradle -Duser.name=Fred helloWorld
:helloWorld SKIPPED
BUILD SUCCESSFUL
```

Plugins in Gradle

- At its core, Gradle provides very little automation
- Most automation, advanced functionality is via plugins
- Example benefits of plugins
  - Promotes reuse
  - Encapsulates imperative logic (allows build scripts to be declarative)
  - More modular approach
- Example actions of plugins
  - Extend Gradle DSL (“the model”)
  - Configure the project based on conventions (i.e. add tasks or sensible defaults)
  - Apply configuration/standards
- Examples additions from plugins
  - tasks
  - domain objects
  - conventions
  - extension of existing objects

38

Types of Plugins in Gradle

- Plugins are build scripts
- Two types:
  - Script Plugin – another build script
    - » another build script - may be used for more configuration
    - » invoke via “apply from: ‘build script name.gradle’
    - » invoke via “apply from: “http://url/build script name.gradle”
  - Binary Plugin
    - » class that implements org.gradle.api.Plugin
    - » contains code that does something - imperative rather than declarative
    - » Must be in build script classpath (built-in ones are there by default)
    - » include via “apply plugin: ‘name’
    - » include via “apply plugin: org.gradle.api.plugins.Name

Differences between Maven plugins and Gradle plugins

- In Maven, a plug-in extends functionality via fine-grained task incorporated into Maven lifecycle
- In Gradle, a plug-in may provide one or more actions by extending tasks, but primarily extends DSL and domain
- Using and extending DSL is preferable to writing custom Groovy code
  - More maintainable
  - Express build actions in a high-level language that is meaningful to your organization

Gradle Plugin Portal

- https://plugins.gradle.org/

https://plugins.gradle.org

Plugins Documentation Forums

gradle

Search Gradle Plugins

search by tag or keyword

Want to include your Gradle plugin here?

Plugin	Latest Version
net.eikehirsch.react A simple plugin to use the react-tools to transform jsx to javascript using gradle. #node #react #javascript #jsx	0.3.1 (02 April 2015)
de.gesellix.docker A Docker plugin for Gradle. Details and usage can be found on the project website or the example project at https://github.com/gesellix-docker/gradle-docker-plugin-example. #docker	2015-04-02T20-49-41 (02 April 2015)
at.rknoll.gradle.questasim	

Example Gradle Build File w/plugins


```
apply plugin:'groovy'
apply plugin:'war'

respositories {
        mavenCentral()
}

dependencies {
        groovy "org.codehaus.groovy:groovy-all:1.8.6"
        providedCompile 'javax.servlet:servlet-api:2.5'
        providedCompile 'javax.servlet.jsp:jsp-api:2.2'

        testCompile "junit:junit:4.9"
}
```

Includes all the Java tasks

Makes new tasks available

build task order
:compileJava
:compileGroovy
:processResources
:classes
:war
:assemble
:compileTestJava
:compileTestGroovy
:processTestResources
:testClasses
:test
:check
:build

"configuration" – providedCompile = not added to deployed war archive

Tasks added by Java plugin

compileJava	All tasks which produce the compile classpath. This includes the jar task for project dependencies included in the compile configuration.	JavaCompile	Compiles production Java source files using javac.
processResources	-	Copy	Copies production resources into the production classes directory.
classes	compileJava and processResources. Some plugins add additional compilation tasks.	Task	Assembles the production classes directory.
compileTestJava	compile, plus all tasks which produce the test compile classpath.	JavaCompile	Compiles test Java source files using javac.
processTestResources	-	Copy	Copies test resources into the test classes directory.
testClasses	compileTestJava and processTestResources. Some plugins add additional test compilation tasks.	Task	Assembles the test classes directory.
jar	compile	Jar	Assembles the JAR file
javadoc	compile	Javadoc	Generates API documentation for the production Java source, using Javadoc
test	compile, compileTest, plus all tasks which produce the test runtime classpath.	Test	Runs the unit tests using JUnit or TestNG.
uploadArchives	The tasks which produce the artifacts in the archives configuration, including jar.	Upload	Uploads the artifacts in the archives configuration, including the JAR file.
clean	-	Delete	Deletes the project build directory.
cleanTaskName	-	Delete	Deletes the output files produced by the specified task. For example cleanJar will delete the JAR file created by the jar task, and cleanTest will delete the test results created by the test task.

Java plugin – expected default layout

Directory	Meaning
src/main/java	Production Java source
src/main/resources	Production resources
src/test/java	Test Java source
src/test/resources	Test resources
src/sourceSet/java	Java source for the given source set
src/sourceSet/resources	Resources for the given source set

Java plugin – lifecycle tasks

Task name	Depends on	Type	Description
assemble	All archive tasks in the project, including jar. Some plugins add additional archive tasks to the project.	Task	Assembles all the archives in the project.
check	All verification tasks in the project, including test. Some plugins add additional verification tasks to the project.	Task	Performs all verification tasks in the project.
build	check and assemble	Task	Performs a full build of the project.
buildNeeded	build and build tasks in all project lib dependencies of the testRuntime configuration.	Task	Performs a full build of the project and all projects it depends on.
buildDependents	build and build tasks in all projects with a project lib dependency on this project in a testRuntime configuration.	Task	Performs a full build of the project and all projects which depend on it.
buildConfigurationName	The tasks which produce the artifacts in configuration ConfigurationName.	Task	Assembles the artifacts in the specified configuration. The task is added by the Base plugin which is implicitly applied by the Java plugin.
uploadConfigurationName	The tasks which uploads the artifacts in configuration ConfigurationName.	Upload	Assembles and uploads the artifacts in the specified configuration. The task is added by the Base plugin which is implicitly applied by the Java plugin.

Lab 3 – Plugins and Properties

For non-windows line in build.gradle should be

ext.myDir = './output'

Caching

- By default, Gradle caches all compiled scripts for faster response
- Includes build scripts, init scripts, etc.
- .gradle directory holds cached objects; created first time build is run for project
- On subsequent runs, Gradle uses the cached version if script has not changed
- Otherwise, scripts are recompiled and cache is updated
- Can force recompile with –recompile-scripts option

47

Logging in Gradle

- Gradle supports 6 different logging levels

Level	Used for
ERROR	Error messages
QUIET	Important information messages
WARNING	Warning messages
LIFECYCLE	Progress information messages
INFO	Information messages
DEBUG	Debug messages

- Gradle build file and each task have a logger object
- Formats
  - logger.level message OR
  - logger.log LogLevel.LEVEL, message
- Default shows ERROR, QUIET, WARNING, and LIFECYCLE
- Use --info to see info
- Use --debug to see debug (lots of output!)

Logging Example

logDemo.gradle

```
task logDemo << {

        // error logging level
           logger.error 'error: this is an error!'
           logger.log LogLevel.ERROR, 'error: this is an error!'

       // quiet logging level
          logger.quiet 'quiet: this is an example of a quiet message.'
          logger.log LogLevel.QUIET, 'quiet: this is an example of a quiet
message.'

      // warn logging level
         logger.warn 'warn: this is your final warning!'
         logger.log LogLevel.WARN, 'warn: this is your final warning!'

      // lifecycle logging level
         logger.lifecycle 'lifecycle: this is a lifecycle (progress) message.'
         logger.log LogLevel.LIFECYCLE, 'lifecycle: this is a lifecycle (progress)
message.'

      // info logging level
         logger.info 'info: this is an informational message.'
         logger.log LogLevel.INFO, 'info: this is an informational message.'

       // debug logging level
         logger.debug 'debug: lots of output!'
         logger.log LogLevel.DEBUG, 'debug: lots of output!'
}
```

```
C:\Users\sasbcl\playpen>gradle -b logDemo.gradle  logDemo
:logDemo
error: this is an error!
error: this is an error!
quiet: this is an example of a quiet message.
error: this is an example of a quiet message.
warn: this is your final warning!
warn: this is your final warning!
lifecycle: this is a lifecycle (progress) message.
lifecycle: this is a lifecycle (progress) message.
```

```
C:\Users\sasbcl\playpen>gradle -b logDemo.gradle --info  logDemo
Starting Build
Settings evaluated using empty settings script.
Projects loaded. Root project using build file 'C:\Users\sasbcl\playpen\logDemogradle'.
Included projects: [root project 'playpen']
Evaluating root project 'playpen' using build file 'C:\Users\sasbcl\playpen\logDemo.gradle'.
All projects evaluated.
Selected primary task 'logDemo' from project :
Tasks to be executed: [task ':logDemo']
:logDemo (Thread[main,5,main]) started.
:logDemo
Executing task ':logDemo' (up-to-date check took 0.0 secs) due to:
  Task has not declared any outputs.
error: this is an error!
error: this is an error!
quiet: this is an example of a quiet message.
error: this is an example of a quiet message.
warn: this is your final warning!
warn: this is your final warning!
lifecycle: this is a lifecycle (progress) message.
lifecycle: this is a lifecycle (progress) message.
info: this is an informational message.
info: this is an informational message.
:logDemo (Thread[main,5,main]) completed. Took 0.24 secs.
```

```
C:\Users\sasbcl\playpen>gradle -b logDemo.gradle --debug logDemo
09:37:37.968 [INFO] [org.gradle.BuildLogger] Starting Build
09:37:37.968 [DEBUG] [org.gradle.BuildLogger] Gradle user home: C:\Users\sasbcl\.gradle
09:37:37.984 [DEBUG] [org.gradle.BuildLogger] Current dir: C:\Users\sasbcl\playpen
09:37:37.984 [DEBUG] [org.gradle.BuildLogger] Settings file: null
09:37:37.984 [DEBUG] [org.gradle.BuildLogger] Build file:
C:\Users\sasbcl\playpen\logDemo.gradle
09:37:38.046 [DEBUG] [org.gradle.initialization.buildsrc.BuildSourceBuilder] Starting to build the
build sources.
09:37:38.046 [DEBUG] [org.gradle.initialization.buildsrc.BuildSourceBuilder] Gradle source dir
does not exist. We leave.
09:37:38.046 [DEBUG] [org.gradle.initialization.DefaultGradlePropertiesLoader] Found env project
properties: []
…
09:37:42.152 [INFO] [org.gradle.execution.taskgraph.AbstractTaskPlanExecutor] :logDemo
(Thread[main,5,main]) completed. Took 0.162 secs.
09:37:42.152 [DEBUG] [org.gradle.execution.taskgraph.AbstractTaskPlanExecutor] Task worker
[Thread[main,5,main]] finished, busy: 0.162 secs, idle: 0.0 secs
09:37:42.152 [DEBUG] [org.gradle.execution.taskgraph.DefaultTaskGraphExecuter] Timing:
Executing the DAG took 0.178 secs
09:37:42.162 [LIFECYCLE] [org.gradle.BuildResultLogger]
09:37:42.162 [LIFECYCLE] [org.gradle.BuildResultLogger] BUILD SUCCESSFUL
09:37:42.162 [LIFECYCLE] [org.gradle.BuildResultLogger]
09:37:42.162 [LIFECYCLE] [org.gradle.BuildResultLogger] Total time: 8.731 secs
```

Gradle Wrapper

- Automatic Gradle download
  - No install
  - Overrides installed version if there is one
- Allows projects to build with precise version of Gradle
- By default, downloads from Gradle web site, but can configure to use internal site
- Generates wrapper files that should be checked into source control
- Available options
  - Destination of downloaded distribution zip
  - Destination of unpacked distribution
  - Source URL of distribution zip
- Extended info in Wrapper task documentation

50

Gradle Wrapper

- Steps to use:
  - Add wrapper task into gradle build script

```
task wrapper (type :Wrapper) {
    gradleVersion = '1.0-milestone-6'
}
```

  - Generate wrapper files

```
gradle wrapper
- build.gradle
- gradle
    - wrapper
        - gradle-wrapper.jar ← bootstrap jar (small)
        - gradle-wrapper.properties ← properties for jar
- gradlew ← shell script
- -gradlew.bat ← Windows batch file
```

  - Gradlew build

Default Tasks

- defaultTasks is keyword
- establishes which tasks to run by default (if no others are specified at invocation time)

Dependencies Between Tasks

- Example tasks that depends on other one
- Invoke with gradle –q world

```
task hello << {
        print 'hello, ‘
  }
  task world(dependsOn: hello) << {
        println 'world‘
  }
```

- Can also declare dependencies on tasks that are defined later

```
task world (dependsOn: hello) << {
        println ‘world’
  }
  task hello << {
        print ‘hello, ‘
  }
```

Custom Task Types

- Can define custom tasks if needed in build file
- Extend DefaultTask (or one of its descendents)
- Use Groovy syntax to define properties and methods
- Can also define larger custom tasks in buildSrc directory at project root
- buildSrc directory is automatically compiled and added to build classpath
- See @org.gradle.api.tasks.TaskAction

```
class CheckTask extends DefaultTask {
        String refValue = 'some_value'
        @TaskAction
      def compare() {
                        println refValue
              // more work
        }
  }
  task  checkVal (type: CheckTask)  {
         compare (0755') {
            fileMode = 0755
            include '**/*.java'
        }
}
```

How to add custom code to Gradle

- In the build script itself - in a task action block
- In the buildSrc directory
- In a separate build script file imported into the main build script
- In a custom plug-in written in Java or Groovy

Other Tasks

- Tasks can come from plug-ins
- Tasks are also provided (built-in) to Gradle DSL – as seen in Gradle command-line use

Lab 4 – Enhancing our Build Script and Using the Gradle Wrapper

Dependency configurations

- Dependencies are grouped into “configurations” – a named set of dependencies.
- Specify inside dependences block – i.e. dependencies { }
- Java plugin provides some standard configurations – each represents classpaths that the plugin uses
- compile
  - dependencies needed to compile production source of project
  - used by compile Java task
- runtime
  - dependencies needed by production classes at runtime (includes compile time dependencies by default)
- testCompile
  - dependencies needed to compile test source (includes compiled production classes and compile time dependencies)
  - used by compileTestJava task
- testRuntime
  - dependencies needed to run tests (includes compile, runtime, and test compile dependencies by default)
  - used by test task

Dependency configurations (continued)

- archives
  - artifacts produced by project
  - used by uploadArchives task
- default
  - default configuration used by a project dependency on this project
  - contains artifacts and dependencies required by project at runtime
- For each source set added to the project, Java plugins add dependency configurations below
- sourceSetCompile
  - compile time dependencies for the given source set
  - used by compileSourceSetJava task
- sourceSetRuntime
  - run time dependencies for the given source set

Managing Dependencies

- With Gradle, define dependencies in build file and it handles needed configuration for various tasks
- Maven and Ivy – tools w/ support for dependency management
- In Gradle, dependencies are grouped into configurations
- Configurations …
  - Have a name
  - Can extend each other
  - Optionally a resolution strategy (for version conflicts)
  - Visibility (could choose not to see configuration outside of project)
- Can run gradle –q dependencies to see list

Managing Dependencies - Terms

- Dependency management includes
  - the things the project needs to build and run – dependencies
  - the things the project produces to build and upload - publications
- Dependency resolution
  - You tell Gradle what the dependencies are – it finds them and makes them available
- Transitive dependencies
  - Dependencies of the dependencies – same as above
- Publication
  - You declare the publications of the project; Gradle takes care of building them and publishing them
  - Publishing might be:
    - » Copying files to a local directory
    - » Upload to a remote Maven or Ivy repository
    - » Using files in another project in same multi-project build

External Dependencies

- Usually projects have external dependencies, such as JAR files
- In Gradle, external dependencies (artifacts) are located in a *repository*
- A repository can be used for:
  - Fetching dependencies
  - Publishing artifacts
  - Both
- Examples:

```
repositories {mavenCentral() }
dependencies {
    testCompile group: ‘junit’, name: ‘junit’, version: ‘4.+’
}
```


62

Repository Dependencies

- Repositories
  - hold dependencies
  - have a layout that defines a pattern for path to a versioned module
  - Gradle does not define any repositories by default
- Gradle understands:
  - Maven repositories
  - Ivy repositories
  - Configured custom repositories available via
    - » file system
    - » HTTP
    - » SSH
    - » other protocols

Repository Dependencies (continued)

- Maven example
  - preconfigured – mavenLocal and mavenCentral – known layout

```
repositories {
    mavenCentral()
}
```

  - custom

```
repositories {
    maven(name: ‘IUO repository’) {
        credentials {
// properties defined in $USER_HOME/.gradle/gradle.properties
            username = usernameIUO
            password = passwordIUO
            }
    url = ‘http://intranet/IUOrepo’
    }
}
```

Repository Dependencies (continued)

- Ivy example
  - layout is customizable – can set layout = ‘maven’ or ‘gradle’ or custom with ivyPatterns and artifactPatterns

```
repositories {
    ivy {
        credentials {
// properties defined in $USER_HOME/.gradle/gradle.properties
            username = usernameIUO
            password = passwordIUO
            }
    url = ‘http://intranet/IUOCustom’
    ivyPatterns ‘[module]/[revision]/ivy.xml’
    artifactPatterns ‘[module]/[revision]/[artifact] (.[ext])’
    }
}
```

Local Directory Dependency

- Use flatDir() method
- flatDir accepts arguments or a closure to configure dir
- Resolves files using first match among a set of patterns

```
repositories {
    flatDir (dir: '../lib', name:  'libs directory')
    flatDir {
        dirs '../local-files', '/vol/shared-stuff'
        name = 'my local shared stuff'
    }
}
```

Declaring External Dependencies

- Add inside dependency configuration
- Long form

```
dependencies {
        configuration group: <value>, name: <value>, version: <value>
}
```

- Short form

```
dependencies {
        configuration <group>:<name>:<version>
}
```

- Examples

```
compile group: 'org.hibernate', name: 'hibernate-core', version: '3.6.7.Final'
compile 'org.springframework:spring-aop:3.1.1.RELEASE'
```

Other Common Types of Dependencies

- Project

  compile project (‘ :project2’)

- File

  compile files (‘abc.jar’, ‘xyz.jar’)

  compile fileTree (dir: ‘sub1’, include: ‘*.jar’)

- Module – overrides repository descriptor files

  compile module(group:name:version)

Dynamic Versions in Dependencies

- Pertains to last argument for version in dependency specification
- +
  - minimum or greater
- [,]
  - [at front is >=
  - [at end is <
  -] at front is >
  -] at end is <=
- Examples: 2.3.+ [1.0, 2.0]]1.0, 2.0[(, 2.0] [1.0,)
- latest.integration

Dealing with Version Conflicts

- In a version conflict situation, Gradle's default is to use the newest version.
- To override default, set resolutionStrategy property of a dependency configuration.
- Options (applies to transitive dependencies also):
  - failOnVersionConflict()
  - force(group:name:version)
  - eachDependency(rule) – add resolve rule, triggered for each

```
configurations.all {
        resolutionStrategy {
                failOnVersionConflict()
        }
}
```

Publishing

- Plugins usually define what needs to be published
- Need to tell Gradle where to publish artifacts
- Done by attaching a repository to uploadArchives task

```
uploadArchives {
        repositories {
                ivy {
                    credentials {
                      username = “name”
                      password = “pass”
                    }
                url = ‘http://intranet/IUOCustom’
                }
        }
}
```

- Note: Can define an artifact using an archive task

```
    artifacts {
                archives someFile (or jar)
              }
```

Lab 5 – Dependencies and Publishing

Testing in Gradle (for Java)

- Gradle supports JUnit and TestNG
- java plugin provides:
  - additional tasks to compile and run tests
  - additional dependency configurations: testCompile & testRuntime
- must add JUnit as a dependency and repositories
- if following configuration conventions of Gradle, simple script

```
apply plugin: ‘java’
repositories {
        mavenCentral()
}
dependencies {
        testCompile ‘junit:junit:[4.10,)’
}
```

Testing in Gradle (continued)

- To run tests, just invoke gradle with test target
  - gradle test
  - gradle test –info (for extra logging)
- Gradle produces an HTML file that reports the test results
  - index.html in build/reports/test
  - click method name of test to see details
- JUnit is default in Gradle; to switch
  - test.useTestNG()
- Tests are run in separate JVM – can change properties
- Tests can be run in parallel

Test Directory Structure

```
Top-level
└────src
    ├────main
    │        └────java
    │                 └────com…
    └────test
             └────java
                      └────com…
```

- Note: If you put tests outside of test subtree, can get errors because using compileJava environment instead of compileTestJava environment

75

Options for the Test Task

- Define within test {} or “test.”
- enable TestNG support (default is JUnit)

```
useTestNG()
```

- Set a system property for the test JVM(s)

```
systemProperty 'some.prop', 'value
```

- Explicitly include or exclude tests

```
include 'org/foo/**'
exclude 'org/bar/**'
```

- Display standard out and standard error of the test JVM(s) on the console

```
testLogging.showStandardStreams = true
```

- Display basic information about test runs in the console

```
testLogging {
        events 'started', 'passed'
        }
```

- Display targeted information about test failures without noise (such as with -info)

```
testLogging {
        exceptionFormat = ‘full’
        }
```


76

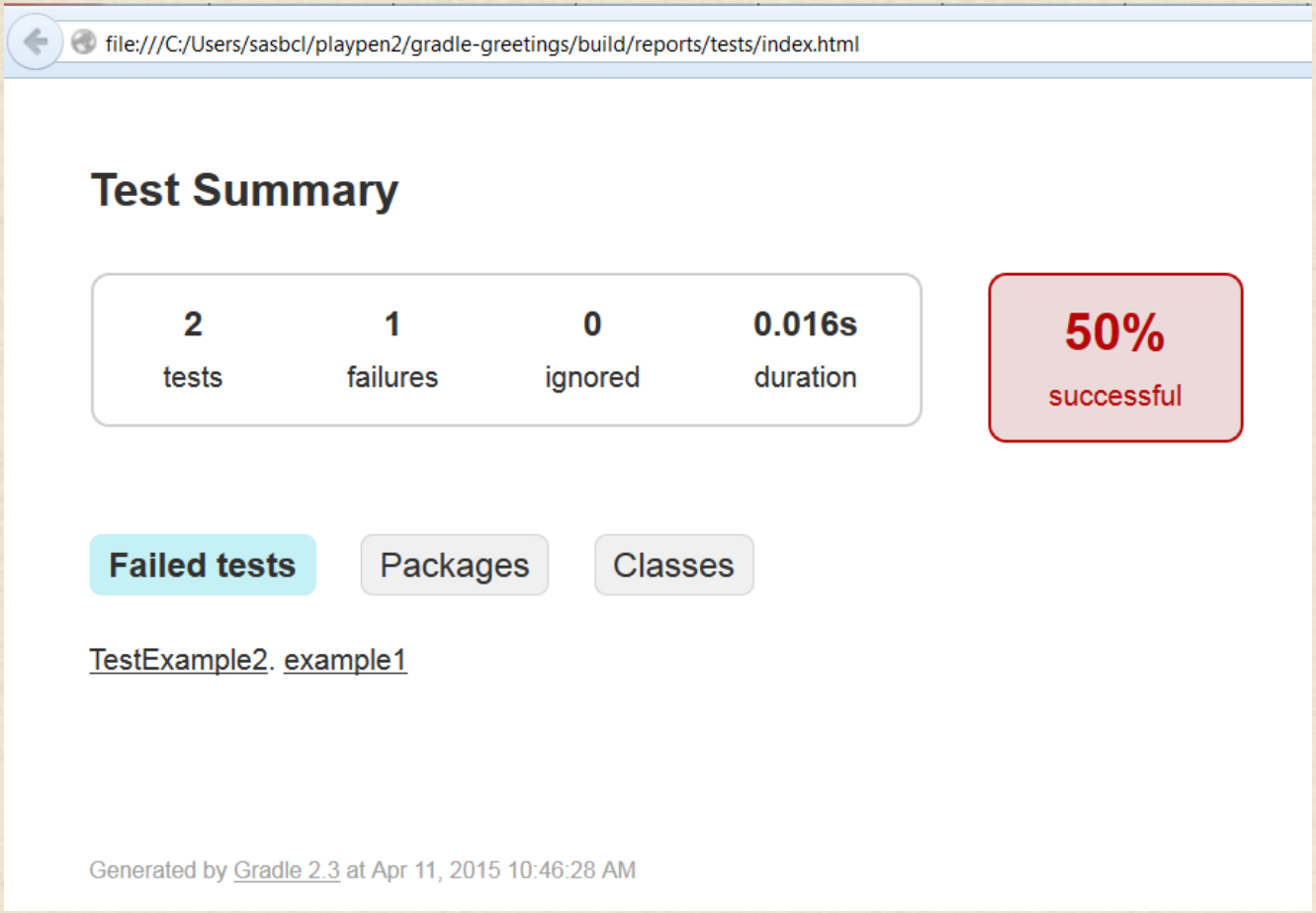
Executing Tests

- Use “test” task to run all tests
- To run a single test, use a java system property:
  - -Dtest.single=<test name> test
- Can use patterns (wildcards) to run sets of tests
  - -Dtest.single=<pattern> test

77

Gradle Testing Web Output

- Gradle creates output reports for testing in <output dir>/reports/test/index.html



Gradle Testing Web Output (2)

- Additional details available via drilling in

file:///C:/Users/sasbcl/playpen2/gradle-greetings/build/reports/tests/classes/TestExample2.html

Class TestExample2

all > default-package > TestExample2

1	1	0	0.011s
tests	failures	ignored	duration

0%
successful

Failed tests Tests

example1

```
org.junit.ComparisonFailure: expected:<[Testing with Gradle] is easy> but was:<[Gradle testing] is easy>
        at org.junit.Assert.assertEquals(Assert.java:125)
        at org.junit.Assert.assertEquals(Assert.java:147)
        at TestExample2.example1(TestExample2.java:10)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:606)
        at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(FrameworkMethod.java:45)
        at org.junit.internal.runners.model.ReflectiveCallable.run(ReflectiveCallable.java:15)
        at org.junit.runners.model.FrameworkMethod.invokeExplosively(FrameworkMethod.java:42)
        at org.junit.internal.runners.statements.InvokeMethod.evaluate(InvokeMethod.java:20)
        at org.junit.runners.ParentRunner.runLeaf(ParentRunner.java:263)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:68)
        at org.junit.runners.BlockJUnit4ClassRunner.runChild(BlockJUnit4ClassRunner.java:47)
        at org.junit.runners.ParentRunner$3.run(ParentRunner.java:231)
        at org.junit.runners.ParentRunner$1.schedule(ParentRunner.java:60)
        at org.junit.runners.ParentRunner.runChildren(ParentRunner.java:229)
        at org.junit.runners.ParentRunner.access$000(ParentRunner.java:50)
        at org.junit.runners.ParentRunner$2.evaluate(ParentRunner.java:222)
        at org.junit.runners.ParentRunner.run(ParentRunner.java:300)
```

Testing: Miscellaneous

- Test is a type of Task and can be inherited from

```
task MyTest(type: Test, dependsOn: testClasses) {
    include '**/*UnitTest*'
}
```

- This also allows for exercising the same api's as Test

```
gradle -DMyTest.single=U* MyTest
```

- Can apply desired properties to all tests

```
tasks.withType(Test) {
        testLogging {
                events  'started', 'passed'
        }
}
```


80

Lab 6 - Testing with Gradle

Source Sets

- Source set - a collection of source files that are compiled and executed together
- Used to group together files with a particular meaning in a project without having to create another project
- In Java plugin, two included source sets
  - main (default for tasks below – assumed if <SourceSet> omitted)
  - test
- Plugin adds 3 tasks for source sets
  - compile<SourceSet>Java
  - process<SourceSet>Resources
  - <SourceSet>Classes
- Source sets also provide properties such as
  - allSource – combination of resource and Java properties
  - java.srcDirs – directories with Java source files

Source Sets (continued)

- Can define your own source sets via sourceSets property of the project

```
sourceSets {
    externApi
}
```

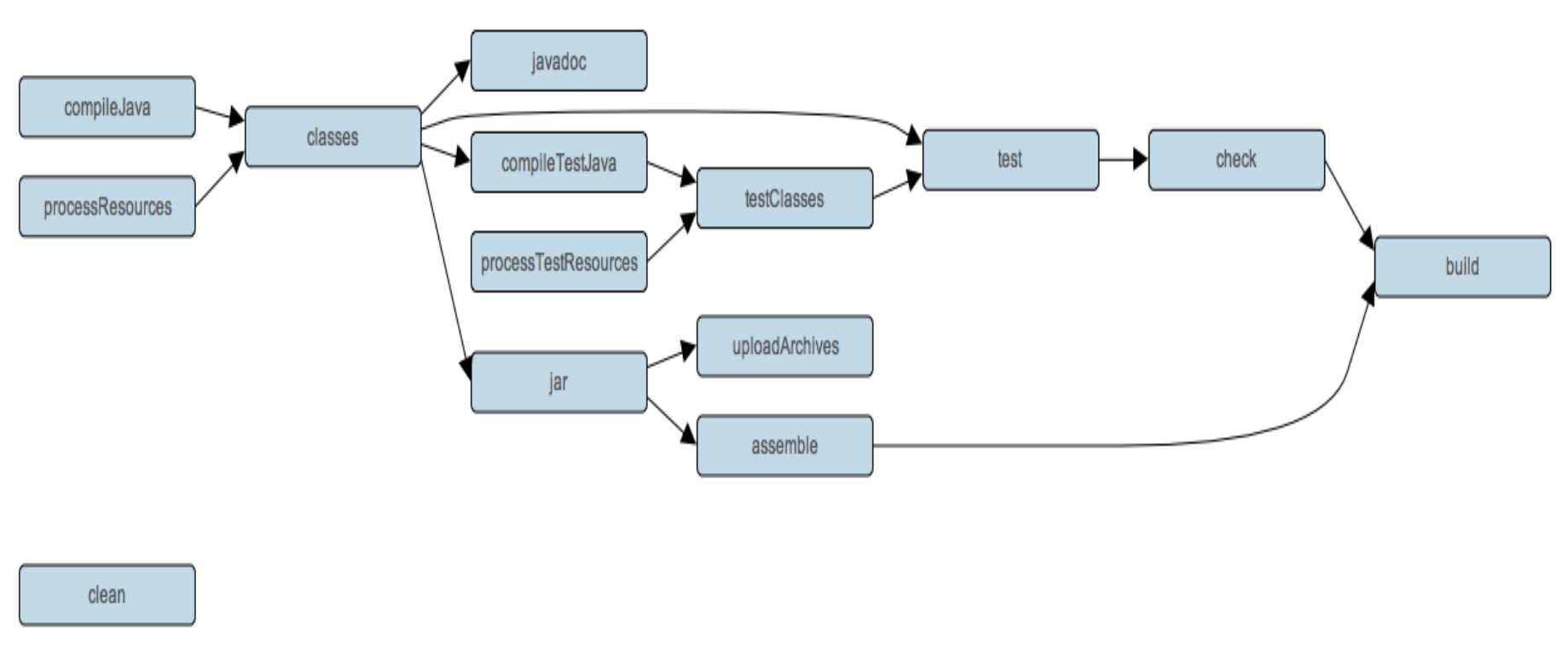
- Get 3 new tasks: externApiClasses, compileExternApiJava, and processExternApiResources
- Custom configuration (also how you would define structure for new ones)

```
sourceSets {
    main {
        java {
            srcDir 'src/java'
        }
        resources {
            srcDir 'resources/java'
        }
}
```

Source Set Properties

Property name	Type	Default value	Description
`name`	`String` (read-only)	Not null	The name of the source set, used to identify it.
`output`	`SourceSetOutput` (read-only)	Not null	The output files of the source set, containing its compiled classes and resources.
`output.classesDir`	`File`	*`buildDir`*`/classes/`*`name`*	The directory to generate the classes of this source set into.
`output.resourcesDir`	`File`	*`buildDir`*`/resources/`*`name`*	The directory to generate the resources of this source set into.
`compileClasspath`	`FileCollection`	`compile`*`SourceSet`* configuration.	The classpath to use when compiling the source files of this source set.
`runtimeClasspath`	`FileCollection`	`output` + `runtime`*`SourceSet`* configuration.	The classpath to use when executing the classes of this source set.
`java`	`SourceDirectorySet` (read-only)	Not null	The Java source files of this source set. Contains only `.java` files found in the Java source directories, and excludes all other files.
`java.srcDirs`	`Set<File>`. Can set using anything described in Section 16.5, "Specifying a set of input files".	`[`*`projectDir`*`/src/`*`name`*`/java]`	The source directories containing the Java source files of this source set.
`resources`	`SourceDirectorySet` (read-only)	Not null	The resources of this source set. Contains only resources, and excludes any `.java` files found in the resource source directories. Other plugins, such as the Groovy plugin, exclude additional types of files from this collection.
`resources.srcDirs`	`Set<File>`. Can set using anything described in Section 16.5, "Specifying a set of input files".	`[`*`projectDir`*`/src/`*`name`*`/resources]`	The source directories containing the resources of this source set.
`allJava`	`SourceDirectorySet` (read-only)	`java`	All `.java` files of this source set. Some plugins, such as the Groovy plugin, add additional Java source files to this collection.
`allSource`	`SourceDirectorySet` (read-only)	`resources + java`	All source files of this source set. This include all resource files and all Java source files. Some plugins, such as the Groovy plugin, add additional source files to this collection.

Task Relationships from the Java Plugin



Lab 7 – Working with Source Sets

The Gradle Daemon

- Gradle has a daemon process
- Decreases startup time
- To use, create a gradle.properties file in the appropriate area:
  - /home/<username>/.gradle/ (*Linux*)
  - /Users/<username>/.gradle/ (*Mac*)
  - C:\Users\<username>\.gradle (*Windows*)
- With this line:
  - org.gradle.daemon=true
- Could also create gradle.properties file at root of project
- Daemon dies off after 3 hours of inactivity
- Note: Anything using the Gradle tooling api (eclipse/intelliJ integration already uses the daemon)
- Can also add other properties in properties file:
  - org.gradle.jvmargs=-Xmx2048m -XX:MaxPermSize=512m -XX:+HeapDumpOnOutOfMemoryError -Dfile.encoding=UTF-8
- Command line --daemon invokes build with daemon explicitly
- Command line --stop stops the daemon

Gradle Continuous Build Functionality

- New as of Gradle 2.14
- Works by keeping Gradle running and monitoring for changes in inputs
- Invoke by passing -t or --continuous option when starting Gradle
- Runs until stopped by keyboard interrupt
- Modifying and saving an input file triggers Gradle to rebuild (thus continuous build)

More (Potentially) Useful Gradle Command Line Options

- -?, -h, --help
  - Shows help message
- -a, --no-rebuild
  - Don’t rebuild project dependencies
- -c, --continue
  - Continue task execution after a failure
- --gui
  - Launches a Gradle Gui
- -m, --dry-run
  - Runs build with all task actions disabled

89

More (Potentially) Useful Gradle Command Line Options

- --parallel (incubating)
  - Build the projects in parallel. Gradle will attempt to figure out optimum number of threads to use.
- --profile
  - Profiles build execution times. Puts report in buildDir/reports/profile directory
- --recompile-scripts
  - Forces recompiling of cached build scripts
- --refresh-dependencies
  - Refresh state of depenendencies

90

More (Potentially) Useful Gradle "Built-in" Gradle Tasks

- [project]: implies a specific project or none
  - If no [project], defaults to root project - denoted by ":"
- [project]:components - Displays components produced by a project [incubating]
- [project]:dependencies - Displays all dependencies declared in project
- [project]:dependencyInsight - Displays insight (details) into a specific dependency in project
- [project]:projects - Displays the sub-projects of project

91

Lab 8 – Using the continuous build feature, working with the daemon, and using more command line options.

Multi-project Builds

- Gradle has very good support for multi-project builds
- Easy to configure multiple projects
- Gradle can resolve dependencies between projects
- Gradle will build needed projects in the right order

Multi-project Builds – Configuration

- Configuration
  - Configuration of all projects happens before any task is executed
  - So even when invoking just a single task, all the projects of a multi-project build are configured first
  - Not the most efficient model in all cases (configuration time may be a factor)
- Configuration on demand
  - Special mode that attempts to configure only projects that are relevant for the requested task(s)
  - Eventually will become the default mode
  - Can be configured for every build via gradle.properties
  - Can be configured for just the current build via command line

Multi-project Builds – Execution

- Project Paths
  - Can execute any task from any project
  - Path of a task is the name of the project and a colon (:) followed by the task name
  - Root project is denoted by just a colon – has no specific name
- Determining whether a project should be executed as a single or multi-project build
  - Looks for settings.gradle file in directory named “master” at same level as current directory
  - If no settings.gradle file found, searches parent directory for a settings.gradle file
  - If settings.gradle is not found, project is executed as a single project build
  - If settings.gradle file is found, and current project is included in the multi-project definition, project is executed as part of multi-project build. Otherwise, executed as single project build.

95

Settings.gradle

- File in root project directory
- Defines what projects make up the multi-project build
- Hierarchial layout
  - include ‘sub1’, ‘sub2’

```
root/
    sub1/
    sub2/
```

- Flat layout
  - Create ‘master’ directory in root project directory
  - Put build.gradle and settings.gradle in ‘master’
  - Change ‘include’ to ‘includeFlat’
  - Projects are referenced relative to parent directory of master

    i.e. master/../tree

Using a single build.gradle file

- For a multi-project build, can define settings for all projects in the root build.gradle file
- Reference a project via the project{} method
- Use complete name of the project as the argument
- Additional method – allprojects{}
  - allows applying project tasks and properties to all projects
- Additional method – subprojects{}
  - only tasks and properties of subprojects are configured

97

Lab 9 – Multi-project Builds

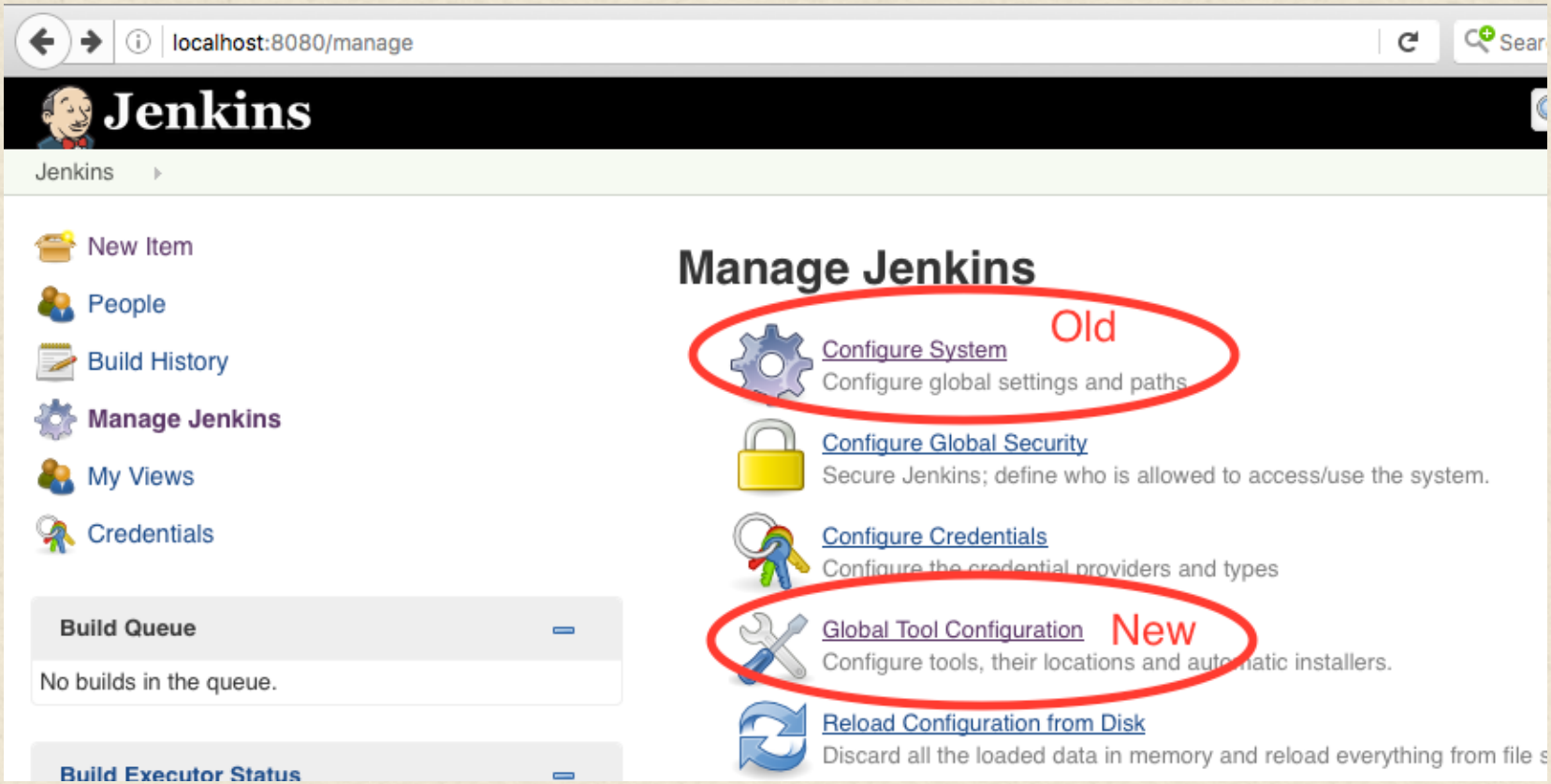
Gradle Plugin for Jenkins

- https://wiki.jenkins-ci.org/display/JENKINS/Gradle+Plugin
- Makes it possible to invoke a Gradle build script as the main build step
- Install through the usual way - Manage Plugins in Jenkins (or HPI file)
- Note: May already be installed in newer versions of Jenkins
- Jenkins provides ways to
  - Specify tasks
  - Specify root project (if multi-project build)
  - Download artifacts
  - View test results

99

Configuring the Gradle Installation

- Old – Manage Jenkins
- New – Global Tool Configuration



Configuring the Gradle Installation

- Setting the global Gradle configuration

localhost:8080/configureTools/

Jenkins › Global Tool Configuration

git

Install automatically

Delete Git

Add Git

description

Gradle

Gradle installations

Gradle

name: Gradle 2.14

GRADLE_HOME: /usr/local/gradle/gradle-2.14/bin

Install automatically

Delete Gradle

Add Gradle

List of Gradle installations on this system

Ant

Ant installations

Add Ant

Save Apply

Creating a Job to use the Gradle Plugin

- Create a new free-style job

localhost:8080/view/All/newJob

Jenkins

Jenkins ▸ All ▸

New Job
People
Build History
Manage Jenkins
Credentials

Build Queue
No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Job name Gradle-Greetings

Build a free-style software project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM w

Build a maven2/3 project
Build a maven 2/3 project. Jenkins takes advantage of your POM files and drastically reduc

Build multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing c

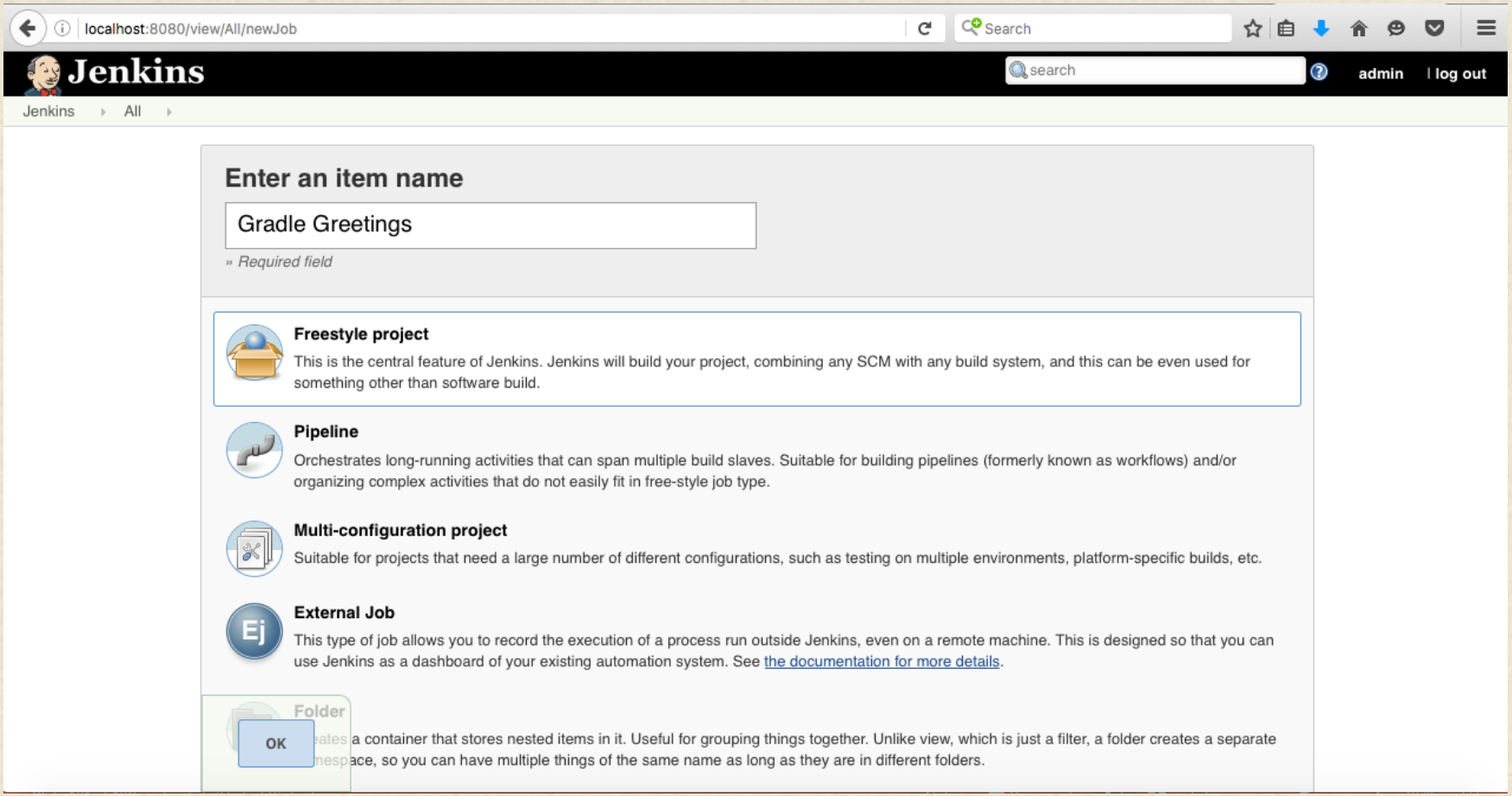
Monitor an external job
This type of job allows you to record the execution of a process run outside Jenkins, even existing automation system. See the documentation for more details.

OK

Help us localize this page

Creating a Job to use the Gradle Plugin

- Create a new Freestyle project



103

Creating a Job to use the Custom Workspace

- Point to our existing workspace

localhost:8080/job/Gradle-Greetings/configure

Jenkins ▸ Gradle-Greetings ▸ configuration

Delete Project

Configure

Build History (trend)

#3 Apr 12, 2015 4:03:34 PM

#2 Apr 12, 2015 4:00:11 PM

#1 Apr 12, 2015 3:54:46 PM

RSS for all RSS for failures

Discard Old Builds

This build is parameterized

Disable Build (No new builds will be executed until the project is re-enabled.)

Execute concurrent builds if necessary

Advanced Project Options

Quiet period

Retry Count

Block build when upstream project is building

Block build when downstream project is building

Use custom workspace

Directory C:\users\retstudent\playpen\gradle-greetings

Display Name

Source Code Management

CVS

CVS Projectset

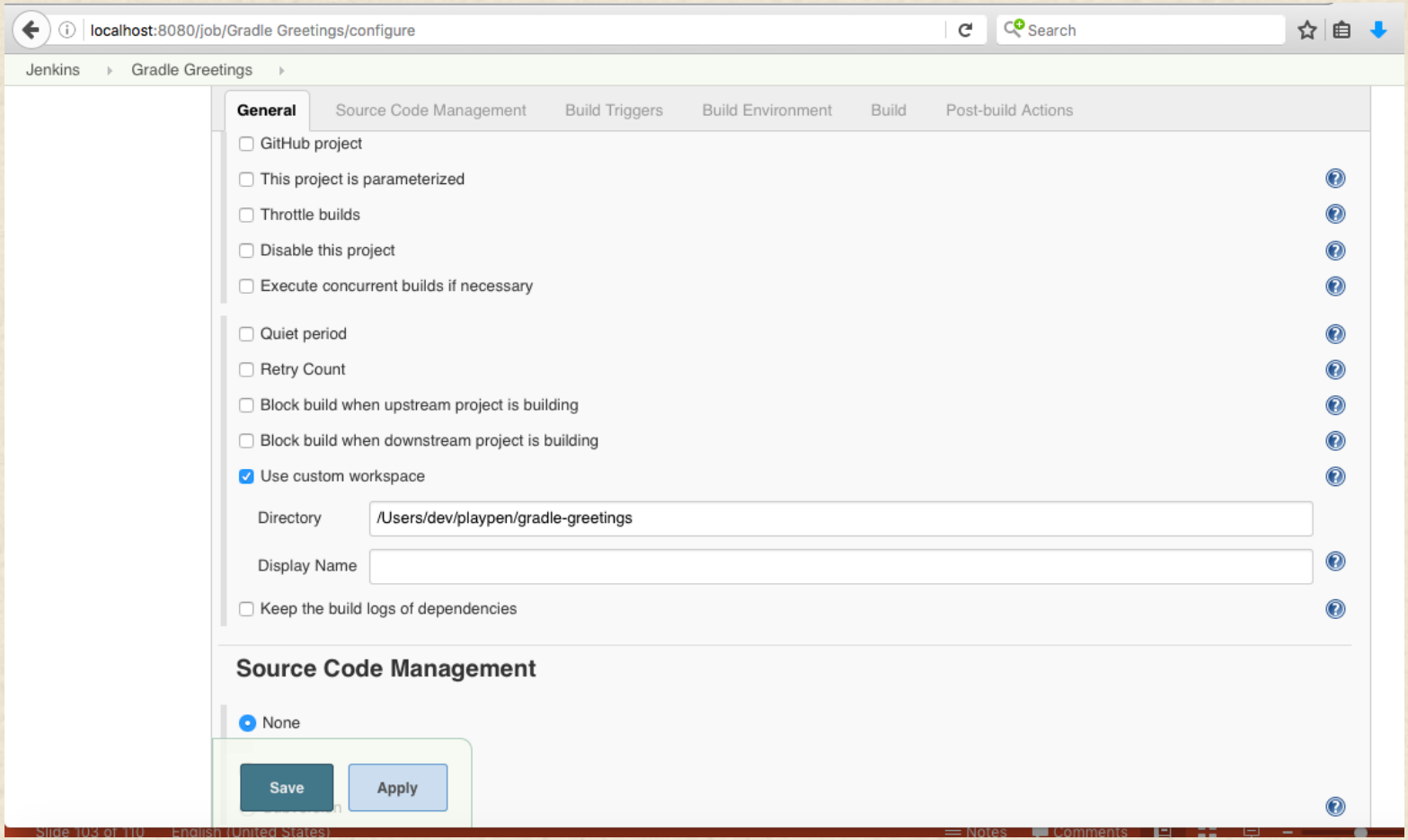
None

Subversion

Save Apply

Creating a Job to use the Custom Workspace

- Point to our existing workspace



Configuring the Build to use Gradle

- Add build steps for Gradle

localhost:8080/job/Gradle-Greetings/configure

Jenkins ▸ Gradle-Greetings ▸ configuration

☐ Build after other projects are built

☐ Build periodically

☐ Poll SCM

Build

Invoke Gradle script

◉ Invoke Gradle

Gradle Version (Default)

○ Use Gradle Wrapper

Build step description

Switches

Tasks clean build

Root Build script

Build File

Specify Gradle build file to run. Also, some environment v

Force GRADLE_USER_HOME to use workspace ☐

Add build step

Configuring the Build to use Gradle

- Add build steps for Gradle

localhost:8080/job/Gradle Greetings/configure

Jenkins ▸ Gradle Greetings ▸

General | Source Code Management | Build Triggers | Build Environment | **Build** | Post-build Actions

Build

Invoke Gradle script

Invoke Gradle

Gradle Version: (Default)

Use Gradle Wrapper

Build step description: Simple gradle integration demo

Switches:

Tasks: clean build -x test

Root Build script:

Build File:

Specify Gradle build file to run. Also, some environment variables are available to the build script

Force GRADLE_USER_HOME to use workspace

Save | Apply

Configuring the Build to use Gradle

- Add post-build steps for Gradle

localhost:8080/job/Gradle-Greetings/configure

Jenkins ▸ Gradle-Greetings ▾ ▸ configuration

Build File

Specify Gradle build file to run. Also, some

Force GRADLE_USER_HOME to use workspace

Add build step

Post-build Actions

Archive the artifacts

Files to archive output/libs/*.jar

Publish JUnit test result report

Test report XMLs output/test-results/*.xml

Fileset 'includes' setting that specifies the generated raw XML report files, su

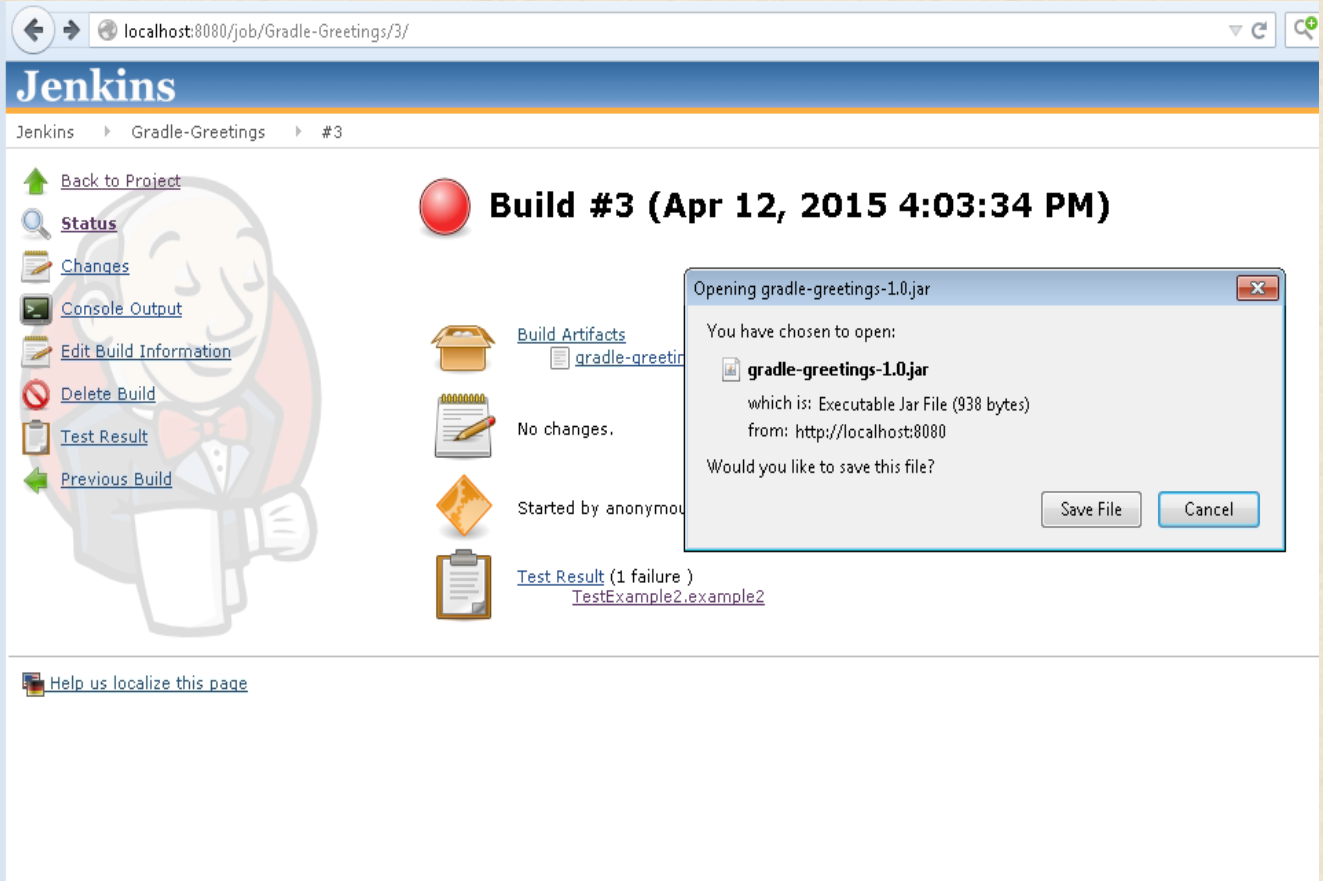
Retain long standard output/error

Add post-build action

Save Apply

Jenkins Functionality with Gradle

- Downloading artifacts



109

Jenkins Functionality with Gradle

- Viewing test results

localhost:8080/job/Gradle-Greetings/3/testReport/

Jenkins

Jenkins ▸ Gradle-Greetings ▸ #3 ▸ Test Results

DISABLE AUTO REFRESH

- Back to Project
- Status
- Changes
- Console Output
- Edit Build Information
- History
- Test Result
- Previous Build

Test Result

1 failures

2 tests
Took 0 ms.
add description

All Failed Tests

Test Name	Duration	Age
TestExample2.example2	0 ms	1

All Tests

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
(root)	0 ms	1	+1	0		1	+1	2	+2

Help us localize this page

Page generated: Apr 12, 2015 4:06:54 PM REST API Jenkins ver. 1.548

Optional Lab 10 – Using Gradle with Jenkins

That’s all - thanks!