Gradle Training Exercises

01-Set Up

1. Extract `gradle-x-all.zip` from the course dir to `C:\gradle-x` or `~/gradle-x`.
2. Add an env variable `GRADLE_HOME` pointing to the extracted directory.
3. Add `GRADLE_HOME/bin` to the `PATH` env variable
4. Open Terminal and execute: `gradle -v`
5. Check out: `http://gradle.org/downloads.html`
6. Extract `groovy-binary-1.7.8.zip` from the course dir to `C:\groovy-1.7.8` or `~/groovy-1.7.8`
7. Add an env variable `GROOVY_HOME` pointing to the extracted directory.
8. Add `GROOVY_HOME/bin` to the `PATH` env variable
9. Open Terminal and execute: `groovy -v`

> The Groovy setup is for the workshop labs… The Gradle download has everything it needs to function on its own.

02-Quickstart

1. Go to: `GRADLE_HOME/samples/java/quickstart`.
2. Check the output of `gradle help`, `gradle -?` and `gradle tasks`.
3. Build the archives for this Java project and try to find them.
4. Run the tests of this project with the Gradle UI. Start the UI with: `gradle --gui`
5. Open and Examine the `build.gradle` of the `quickstart` project.
6. Run `gradle dependencies`

03-Tasks

1. Add a `hello` task that prints 'hello world'.
2. Execute this task.
3. Add a `date` task that prints out the current date.
4. Execute this task.

04-Task Dependencies

1. Make the `date` task depend on the `hello` task.
2. Execute the `date` task.
3. Execute `gradle tasks --all`.
4. The `--dry-run` (or `-m`) command line option executes the build but disables all actions. Execute `date` with the `dry-run` option
5. Add some top level `println` statements to the script.
6. Add a `println` statement to the configuration closure of the `date` task (create one if you haven't used one yet).
7. Execute the `hello` task and analyze the output.

05-Groovy

1. Convert a list of numbers [0, 32, 70, 100] from fahrenheit to celsius. Use a GString to print the values.

   > conversion is f = c * 9/5 + 32

2. Print the IP Address(es) of the machine your are running on.

06-Applying Plugins

1. Go to: `GRADLE_HOME/samples/java/quickstart`.
2. Change the apply argument for the Java plugin from the id to the type (`org.gradle.api.plugins.JavaPlugin`) and execute `gradle tasks`
3. Go to the exercise directory.
4. Apply the plugin `info.gradle` in the `plugins` directory to the build. Execute `gradle tasks` to see what task has been added. Execute the task.
5. Apply the plugin `https://github.com/kensipe/gradle-samples/raw/master/poznan.gradle` to the build. Execute `gradle tasks` to see what task has been added. Execute the task.

07-Testing

1. Run the tests with `testReport` set to false. What do you see?
2. Run the tests with different settings for `forkEvery`. What do you see?
3. Run the tests with different settings for `maxParallelForks`. What do you see?
4. Add a listener that, if a test fails, opens the test results xml.

   Hint:

   1. The xml test result can be found at `build/test-results/TEST-<test-class-name>.xml`.
   2. To execute an external command in Groovy you can do "command arg".execute()
   3. Check the javadoc of the `org.gradle.api.tasks.testing.Test.afterTest` method to learn about its arguments.

08-Ant

1. Execute `ant -projecthelp` to understand what is possible
2. Examine the build.xml file
3. Execute the `ant hello` task
4. In the Gradle file, import the ant file with `ant.importBuild 'build.xml'`
5. Execute the ant hello task through gradle with `gradle hello`
6. Remove the intro target in the ANT file
7. Execute the `ant hello` task (notice the failure)
8. Create a task in Gradle called `intro` providing a message at execution of "Hello from Gradle"
9. Execute the ant hello task through Gradle with `gradle hello`
10. Extend the hello task in gradle with a message. (hint: `hello.doFirst {}`)
11. Execute the ant hello task through gradle with `gradle hello`

09-Dependencies

1. Add the maven central repository and a configuration named `mydeps`. Assign the `org.apache.httpcomponents:httpclient:4.0.3` dependency to `mydeps`.
2. Add a task `showDeps` that prints out the files of the `mydeps` configuration.
3. Add task `copyDeps` that copies the files of the `mydeps` configuration into the `build/deps` dir.
4. Execute `gradle dependencies`

10-Web-Project

1. This is a fully functional Spring MVC project for which you will create the build file.
2. First apply the plugin "jetty" with `apply plugin:'jetty'`
3. Add the 2 needed repositories: `mavenCentral()` and

`http://maven.springframework.org/milestone`

mavenRepo urls: 'http://maven.springframework.org/milestone'

4. Add the required dependencies for compile: 'org.springframework:spring-webmvc:3.0.0.RELEASE'
5. Add the required test dependencies: 'org.springframework:spring-test:3.0.0.RELEASE' and 'junit:junit:4.7'
6. Execute `gradle jRW`

   This is shorthand for jettyRunWar
7. Open a browser to `http://localhost:8080/hellog/helloWorld.html`

11-Multi-Project Builds

1. Investigate the structure of the multiproject build. Execute the `build` task from the root project and observe what is happening.
2. Go to the `api` project. Execute `build` from there. Execute also `buildNeeded` and `buildDependent`. What is different compared to executing the build task.
3. Execute the `build` task of the `api` project from the root project directory.
4. Execute `gradle projects` from the root directory.
5. Execute `gradle projects` and `gradle :projects` from the services directory.
6. Execute `gradle tasks` and `gradle :api:tasks` from the root project directory.
7. Execute `gradle :services:webservice:properties` from the root project directory.
8. Execute `gradle --profile clean build` from the root project. Have a look at the profile report in: `build/reports/profile`