Version Control and Build Management Lab

When working in larger teams, it becomes imperative to manage the source code as multiple developers work on it. Additionally, it is important to have a stable, automated build cycle that does not require any human intervention to accomplish. The ultimate goal would be one simple command that would perform the complete build and all associated actions.

This lab will also introduce you to creating an executable jar file, linking a third party library to your build, and the use of packages.

Version Control

There are two primary types of version control systems in use: central repository and distributed. Up until recently, the lab was always based on a central repository versioning system called Subversion. Subversion is still used in many places, but this semester, for M2, we will use a distributed versioning system called Git. After this milestone, for the remainder of the project, you may use the system of your choice.

Git is described in http://git-scm.com/doc

There is a nice interactive tutorial for git here: https://try.github.io/levels/1/challenges/1

For central project hosting, you may use any service you wish, although github (https://github.com/) or (https://github.gatech.edu) is the most popular.

Setup

While this part is not going to be graded directly, you will not be able to complete the rest of the lab if you omit it.

Setup Git on your machine with the desired client (git for command line, TortiseGit for file manager integration, the git software from github or EGit for Eclipse integration). Note that for this semester if you are using Netbeans, Andriod Studio, or Intelli-J, those already have Git support out of the box -- although you may need to install the binary distribution of git for the IDE to use.

Note

You should name your repository "repository" (if running your own copy of git) If using remote hosting like GitHub, then just provide us the URL and we will check your code out from there.

Initialize a new Git repository. Be sure that your teammates can access the repository also. You will need to work together to complete some portions of this lab. This step is not required if using GitHub hosting.

Download the M2 resource file (M2.zip) and unzip it.

Import the unzipped files (add, commit and push) as a new project named M2 (Only one person has to do this). From an IDE, select Share Project on Github.

Note that the lib directory contains a resources.jar file. This is to simulate a 3d party library you have to link to. You will need to ensure during the lab that this .jar is on the classpath for compilation and execution.

Create a branch at this point called original. The purpose of this branch is to retain an original unmodified set of code.

Each team member should clone out the project M2.

BE SURE THAT original REMAINS AS THE UNMODIFIED VERSION and that changes to the repository are done on the main (master) branch.

Working with Git

You should work with your Git repository for a bit and in doing so you should complete all of the following tasks at least once:

Edit and commit files as detailed below.

If you examine the src directory, you will see the files Person1.java Person2.java Person3.java Person4.java Person5.java (located in the edu.gatech.oad.antlab.person package). If you have 6 people then just duplicate one one of the code files and rename it Person6.java.

Each person on the team should choose ONE of these so that everyone is working on a different file.

Each person should create a branch named for their name (bob, sally, freda…). Change to this branch.

Now examine your individual file and complete the required method. The javadoc comment will explain what has to be done.

Commit your changes and push back to Git. Merge your code from your development branch into th4e master branch.

Now each team member should edit the file edu.gatech.oad.antlab.pkg1.AntLabMain.java at the line indicated. You are basically sending your name to the constructor of the class you edited previously. Since you are all editing the same file, you will need to handle any merge requirements yourselves. You might want to get together and commit files to force a merge so you get some practice doing merges. You will run into them during the project.

Add and remove files

Each team member should add a text file on the master branch to the top level directory labeled readme.pn.txt where the pn would be p1, p2, p3, p4 or p5 based upon which person you are for the lab. The contents of the file can really be anything, but should include your name and email. Each team

member should delete the text file useless.pn.txt where pn is p1,p2, p3, p4 or p5 based on your team member number. Do NOT delete the wrong files!!

Use good descriptive Log comments for your changes/commits

Roll back changes by viewing (checkout) your original branch from an earlier step. Verify that none of the changes you have made are in the original branch project version you checked out.

View Git logs. You can do this on github by viewing the blame and graphs provided.

View a diff between two versions of a file

Keep playing with git until you feel comfortable. You will probably be using it extensively for the remainder of the semester.

Build files

~~Each **individual** team member will create a build file using gradle that will allow you to perform key tasks to build the application. Name your file with your gtid.gradle for instance: gburdell3.gradle (DO NOT USE YOUR 900 NUMBER). Store your file in the repository.~~

Each team should produce a gradle build file that will run the M2 code.

~~To run gradle with your file (build.gradle is the default if no file specified) use the -b option, for example: gradle -b gburdell3.gradle~~

You may leave your team file named build.gradle if you wish

First, each team member should install Gradle onto their machine. Instructions are in the gradle manual at: http://www.gradle.org/

The manual and documentation on the gradle site covers most of the features of gradle. For this lab, we will only be using the most basic of Gradle features and the Java plugin.

You will need to create an executable jar for the project. To do that, you will need to create a manifest. If you are unfamiliar with manifests, then see the official tutorial at: http://docs.oracle.com/javase/tutorial/deployment/jar/

The application you are building also uses packages. You will need to understand how to use packages, as all professional Java projects use them extensively. Again, if you unfamiliar with packages, see the official tutorial at: http://docs.oracle.com/javase/tutorial/java/package/index.html

Your build file should support the following tasks:

```
compileJava      compiles and builds the project
compileTest      compiles the unit tests for the project
javadoc          creates the html documentation for the project
clean            removes all the files created by this project
jar              creates the executable jar file for this project
```

The jar task should be the default if no task is designated when running gradle. Otherwise, gradle will execute the task you request, for example: gradle compileJava will compile your java project.

Troubleshooting

Some common problems:

- Can't find class AntLabMain This is usually from your Main-Class entry in the jar manifest. Remember that with packages, you must use the fully qualified name (package_name.class_name) not just the class name by itself.
- Can't find class AntLab31 This class is defined in the resources.jar in your library directory. This is caused by your classpath for compilation or execution not containing a reference to this jar. Remember that the manifest must contain a reference to resources.jar and the path must be relative to your jars location.