Version Control Systems, Build Automation and Continuous Integration – Exercises

Systems Engineering Laboratory 1

Note: Please treat these exercises also as professional work. For example, instead of *asdfg* use more meaningful commit messages like *Add acceleration feature* or *Fix #5* (you can find detailed advice in the How to Write a Git Commit Message and the Git Style Guide posts).

1 Initialization

1. Check on the GitHub website your base repositories initialized by the course administrator
2. Initialize your development environment (start the proposed virtual machine and overview the tooling)

2 Markdown exercises

1. [**local**] Clone your project using the local git tool into your development environment.
2. [**local**] Edit and add some notes to the Readme using the Markdown syntax. Present at least three different formatting in your new content. (Use your development environment to do the changes instead of the Web UI)
3. [**local**] Commit, push your changes and check the results on the Web UI.

3 GitHub Flow Exercises

1. [**web**] Define a new feature (or change request) on your project and create a GitHub *Issue Ticket* to maintain it. (You can "request" a *bugfix* or a new *feature*)
2. [**web**] Create a branch for your issue.
3. [**local**] Pull your project and change to the new branch.
4. [**local**] Do a local build using Gradle. Test the project by executing it.
5. [**local**] Do the changes on your code according to the defined ticket. (You can do changes, build and test the project on your development environment)
6. [**local**] Commit and push your changes to the repository.

7. [**web**] Create a pull request (merge request).
8. [**web**] Check your pull request, check the changes, comment it and if it is ok, then approve and merge it.
9. [**local**] Check the `git log` command to show the change history.

4 Merge conflict

1. Figure out how to generate a merge conflict.
2. [**local**] Create two new branches (`branch-A`, `branch-B`).
3. [**local**] Checkout `branch-A`, edit one line in a file and commit the changes.
4. [**local**] Checkout `branch-B`, edit the same line in the same file on a different way and commit the changes.
5. [**local**] Switch back to the master branch and merge `branch-A` and `branch-B` afterwards. Check the results and if a merge conflict occurs, then resolve it.

5 Travis CI

1. [**web**] Login to *Travis CI* using your existing GitHub account.
2. [**local**] Add and edit `.travis.yml` file in order to prepare your project to build with Travis CI (this is a basic Gradle project, you can use the Gradle basic Travis settings).
3. [**local**] Commit and push your project in order to trigger a new build.
4. [**web**] Check the web, check your build, console output and result. Check the results of the JUnit tests.
5. [**local and web**] Do some changes to generate build error. Commit, push and check the results on the web.
6. [**local**] Fix build error and add one new JUnit tests to your project.
7. [**local and web**] Commit and push to trigger build with JUnit tests and check the results.

6 Add external dependencies

1. [**local**] Implement a tachograph module that records the following values to a single collection. Use the Google Guava library's `Table` class to implement the collection.
   - current time
   - joystick position
   - reference speed
2. [**local**] Go to http://mvnrepository.com or http://search.maven.org/ and search for the `guava` dependency.
3. [**local**] Add it to the Gradle project and update the Gradle project configurations.
4. [**local and web**] Commit and push your changes, check your build.
5. [**local**] Add a basic unit test that checks if the collection has some elements.

7 Finishing the lab

1. Commit and push everything to the repository, check if the build is success and present it to the teacher.