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. Fork the https://github.com/ftsrg-retelab/base project to your GitHub account. If you don't have a GitHub account yet, then please register one, confirm the e-mail address and then you can fork the mentioned project.
2. Initialize your development environment. You will use
   - CLI tools (marked by [**CLI**]) using a VM started in the BME Cloud,
   - any type of IDE or editor (marked by [**IDE**]) on your desktop to edit Java code. You should also have a git client, either built it into the IDE or a CLI version.
   - Browser (marked by [**WEB**]) on your desktop to use GitHub web UI and other online CI tools

2 Markdown exercises

1. [**CLI**] Clone your project using the local git tool into your development environment.
2. [**CLI**] Edit and add some notes to the Readme using the Markdown syntax. Present at least three different formatting in your new content. (Use CLI tooling and text editors to do the changes instead of the Web UI or an IDE)
3. [**CLI**] 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. [**CLI**] Pull your project and change to the new branch.

4. [**CLI**] Do a local manual build using Gradle. Test the project by executing the gradle build.
5. [**IDE**] Pull your project and change to the new branch in your IDE. 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. [**IDE**] Commit and push your changes to the repository.
7. [**CLI**] Pull the changes to the cloud instance and do a local manual build using Gradle. Test the project by executing gradle test task and check the results in the JUnit test result XML reports (in the subproject/build folder).
8. [**WEB**] Create a pull request (merge request).
9. [**WEB**] Check your pull request, check the changes, comment it and if it is ok, then approve and merge it.
10. [**CLI**] Check the `git log` command to show the change history.

4 Merge conflict

1. Figure out how to generate a merge conflict.
2. [**CLI**] Create two new branches (`branch-A`, `branch-B`).
3. [**CLI**] Checkout `branch-A`, edit one line in a file and commit the changes.
4. [**CLI**] Checkout `branch-B`, edit the same line in the same file on a different way and commit the changes.
5. [**CLI**] 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 GitHub Workflows

1. [**WEB**] Check the description about building Java projects with gradle using GitHub Actions
2. [**CLI**] Add and edit `.github/workflows/build.yml` file in order to prepare your project to build with GitHub Actions (this is a basic Gradle project, you can use the Gradle basic settings). Take care on that the project requires Java version 8 and use the right step versions base on the documentation (instead of the @commit hash syntax)
3. [**CLI**] Commit and push your project in order to trigger a new build.
4. [**WEB**] Check the Actions tab on the GitHub webUI, check your build, console output and result. Check the results of the JUnit tests.
5. [**IDE and WEB**] Do some changes to generate build error. Commit, push and check the results on the web.
6. [**IDE**] Fix build error and add one new JUnit tests to your project.
7. [**IDE and WEB**] Commit and push to trigger build with JUnit tests and check the results.

6 Add external dependencies

1. [**IDE**] 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. **[IDE]** Go to http://mvnrepository.com or http://search.maven.org/ and search for the `guava` dependency.
3. **[IDE]** Add it to the Gradle project and update the Gradle project configurations.
4. **[IDE and WEB]** Commit and push your changes, check your build.
5. **[IDE]** Add a basic unit test that checks if the collection has some elements.

7 Implement additional GitHub Actions CI workflow (iMSC)

1. **[IDE or CLI]** Add an additional test and build steps to the workflow
2. **[WEB]** Show the execution on the WebUI and interpret the results

8 Finishing the lab

1. Commit and push everything to the repository, check if the build is successful.
2. Finish your report, add the link of your GitHub project to the cover page and upload it to the Moodle.
3. Ensure that you keep the GitHub repository until the end of the semester.