HÁSKÓLI ÍSLANDS
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild

HBV202G: Software Design and Construction · Spring 2025
Andri Valur Guðjohnsen · Dr. Helmut Neukirchen

Assignment 2 · Due Tuesday 4.2.2025 / Friday 7.2.2025

Formalities:

- No PDF submission in this assignment, rather via Gradescope online web form. Remember to select your other team member in the first submission steps.
- You need an account for a Git remote repository, e.g., GitHub: create one if you do not yet have one. With the first push to GitHub, you will need to provide credentials. For Two-Factor authenticaton, see also slides 2-33/34. Independent from that: if your IDE does not have a proper GitHub integration, you need to create some sort of throw-away password, namely a **personal access token** in GitHub (select scope *repo*) and use that (keep a copy) then as password for your username, e.g., when want to access a GitHub repository with a Git client, see `https://mgimond.github.io/Colby-summer-git-workshop-2021/authenticating-with-github.html`
- This assignment requires two students working together via one shared Git remote repository. If you did not manage to work as a team, then you can simulate a second user who is pushing changes to the remote repository by doing edits directly in the web interface of the remote repository (while working in parallel with your IDE).
- It might be that you get at your first pull an get error and in the output, you see then:

  ```
  hint: You have divergent branches and need to specify how to reconcile them.
  hint: You can do so by running one of the following commands sometime before
  hint: your next pull:
  hint:
  hint:   git config pull.rebase false  # merge (the default strategy)
  ```

  This means, the Git used by VS Code does not know how to merge the pulled changes into your local working copy. In this case, select `git config pull.rebase false` with your mouse and copy that text, then switch in VS Code from the output windows to the terminal window: there, type in the `cd` command to change into the directory of your project and paste the copied git command there and press enter. After that, pull should work.

Objectives: Learn using Git (e.g. resolve conflicts) with a local and remote repository.
All major IDEs have by default a Git integration installed (but IntelliJ IDEA and Visual Studio Code need to have the command line Git installed: `https://git-scm.com/downloads`). Get familiar with the Git integration of your IDE: go during class through a Git tutorial for your IDE (should not take longer than 30 minutes), e.g.:

- Eclipse: `https://eclipsesource.com/blogs/tutorials/egit-tutorial/` (For the section *Cloning Repositories*, you need a remote repository: for this, first create one as described in the step 1. below.)
- IntellJ: To see how the Git command line concepts can be used from the IDE:
  `https://www.jetbrains.com/help/idea/2024.3/using-git-integration.html`

  If you rather want to go through some very intense interactive tutorial: on the Welcome to IntelliJ IDEA screen (i.e. close any open project): *Learn → Start Learning*: *Git.* (Do not do *Interactive rebase* or any later parts.) Note that these lessons use the icons intended for advanced users, but you can later also use the *Git* entry in the main menu and the *Git* entry in the context menu on a file or the whole project.

- Visual Studio Code: To see how the Git command line concepts can be used from the IDE: `https://code.visualstudio.com/docs/sourcecontrol/overview`

Note: when the following items ask, e.g., the first student to do something, the second student should also observe what the first student is doing in order to practise Git.

1. One student in the team creates via the web interface of, e.g., GitHub, a new repository, make it *public*, but keep it empty, i.e., do not add a README file, nor a .gitignore, nor a license.

   Add the other student to the repository (GitHub: → *Settings* → *Collaborators* → *Add people*)
   Question 1: In the Gradescope online submission form: write down the URL of your public GitHub project.

2. Both students: In your IDE, do what corresponds to a `git clone` on the command line, i.e. create a new project from that (so far empty) remote repository: use from the web interface the repository `https` URL ending with: `.git`

3. First student: Add a text file in your IDE project that contains a greeting like *Hello world* in one language. Add this file to version control, commit it locally with a reasonable commit message.

   Add another line with a greeting in another language, commit it locally with a reasonable commit message.
   Question 2: In your IDE, have a look at the log (or history or timeline – but the timeline in VS Code does not show branch names, use in this case the command line `git log`) where you see where HEAD is pointing to but also the main branch in your local repository (maybe and the origin/main branch from the remote repository): to which of these branches is HEAD pointing to?

4. First student: Finally, push your two local commits to the remote repository.
   Question 3: To which of the branches is HEAD now pointing to?

5. Second student (if you work alone: simply edit the file directly via the GitHub web interfaces editor): Pull the change from the remote repository. Add to the file a new line with a greeting in a third language. Commit it first locally with a reasonable commit message and finally push that change to the remote repository.

6. First student: Pull from the remote repository. Make a change to the first line, e.g., turn all to uppercase. Commit and push your changes.

7. Second student (if you work alone: the GitHub web always shows the latest remote repository state; therefore, it is not possible to do it "Without doing a pull". As answer to this question, wait until you have reached step 10. where the same problem occurs from the first student's perspective): Without doing a pull from the remote repository, make a change to the last line, e.g., turn all uppercase. Try to commit and push your changes.
   Question 4: What happens when you try to push your changes and how can you solve that problem so that your changes end up in the remote repository?

8. First student: Pull from the remote repository. Except for the last line, add to the end of all lines a "`!`"

9. Second student: Pull from the remote repository. Except for the first line, add to the end of all lines a "`?`". Commit and push your changes.

10. First student: Try to commit and push your changes.
    Question 5: What happens when you try to push your changes and what happens when try to merge changes from the remote repository into your local file?

11. First student: while your IDE might provide you some interactive tool to resolve the conflict and you are welcome to play around with such a tool, you should also exit that tool so that you can directly see the file in the normal IDE text editor so that you see the conflict markers (`<<<<<`, `=====`, `>>>>>`). Think about a reasonable resolution of the conflict and apply that resolution, mark the conflict as resolved (i.e. stage via Git add), then commit and push.
    Question 6: How do you mark in your IDE a conflict as resolved?

12. Both students: now do something so that the second student experiences a conflict and resolves it and pushes it.
    *Question 7: Describe briefly the steps (In the style "*First student: do xyz. Second Student: do abc*") that you did to make the second student experience a conflict!*

13. Both students: pull to make sure that all have the latest version.

14. Both students: create a `.gitignore` file that prevents you from accidentally adding to version control directories or files that are specific to your IDE. (In general: if all agree to use the same IDE in a project, it makes even sense to add IDE-specific files. In this assignment, we want to cover the more general case that two different IDEs are used.)
    Question 8: What is the content of your `.gitignore` file (simply copy the content)?

15. Voluntary, if you still have time: So far, we worked directly on the main branch. But, some prefer to add their changes first to branches and only once the code is working there, merge it to the main branch.

    Practise using branches: the first student creates branch `feature1`, switches to that branch and applies changes there and commits and pushes that branch. The second student does the same with branch `feature2`. Then each student merges their branch back to the main branch. Doing a pull will reveal whether the changes in the branches lead to conflicts when merged into the main branch.