Introduction

Martin Ledvinka

KBSS

Winter Term 2022

Contents

1 Organization

2 Projects in Semester

3 Git

Organization

Course Info

You will learn how to design, develop and integrate “enterprise” applications.

- Learn enterprise-level technologies
- Work in teams on semestral work
- Use source code management tools
- Prepare overall system design document
- Develop a reasonably sized subset of the designed application
- Integrate your application with another application

Evaluation Overview

Total of **100 points** consists of

- **50 points** for semestral work
  - **5 points** for CP0
  - **15 points** for CP1
  - **30 points** for CP2
- **40 points** for exam test
- **10 points** for seminar tasks

You need to get at least **50% of each item**.

Read carefully evaluation details at
`https://cw.fel.cvut.cz/b221/courses/b6b36ear/hodnoceni`.

Overall Course Info

- No frontend
- Git mandatory
- We will be building an application from scratch, a process similar to your semestral work

Organization

- Seminar attendance is **mandatory**
  - Maximum of 3 absences
  - Consultation seminars are voluntary (marked with a * on CourseWare)
- Graded seminar tasks can be finished at home
  - Submission deadline is always midnight of the Sunday immediately after the seminar at which the task was assigned

Graded Seminar Tasks

- Graded tasks will be assigned at selected seminars
- You must earn **50%** of the **10 points** available for the tasks
- Tasks will be related to the e-shop application we will be building during the semester
- For each task, a set of acceptance criteria will be specified. Points will be awarded based on fulfilment of these criteria
- How does it work?
  1. You will **fork** the e-shop Git repository $B221 - eshop$ (lets call this fork RF)
  2. You will add your seminar's teacher to RF's members
  3. For each seminar, there will be a separate branch in $B221 - eshop$
     - E.g., $b221 - seminar - 02 - task$
  4. You will check out the seminar branch in the local copy of your fork RF
     - See GitHub Help
  5. You will work on the task and push the solution into RF
  6. The teacher will evaluate the solution in RF and award you points

Software Stack

- JDK 8 or newer
- Maven 3
- PostgreSQL 9 or newer
- Git
- HTML 5-compatible web browser
- Java IDE
    - We recommend (and support) IntelliJ IDEA or NetBeans
- REST-compatible client
    - We recommend Postman. But cURL or browser plugins like REST client for Firefox also possible

Task 1 – Installation Check

Verify that you have all the required technologies installed or install them now.

Notes

- Lab computers should have all of the above except Postman.
- IntelliJ IDEA comes with bundled Git and Maven. NetBeans have bundled Git client as well. We still recommend installing both separately.

Projects in Semester

Semestral Work

Overview

The goal of your semestral work is to design a small web application solving some practical use-case. Next, you will choose a part of the application and implement it using technologies you learn in this course. In the end, you will integrate the application with an application of another team/our application/a remote SSO service, build a UI for it, or provide a comprehensive set of REST API tests in Postman.

Notes

- You will work in teams of two
- First checkpoint is in the third week, so do not hesitate to form a team and select a topic
- Topics *should* not repeat in a single seminar group

Example Project

We will be building a simplified e-shop application. The frontend is already prepared, we will be gradually working on the backend. We will

1. Start with project inception
2. Create object model
3. Build persistence layer
4. Write business logic
5. Create web services
6. Secure the application

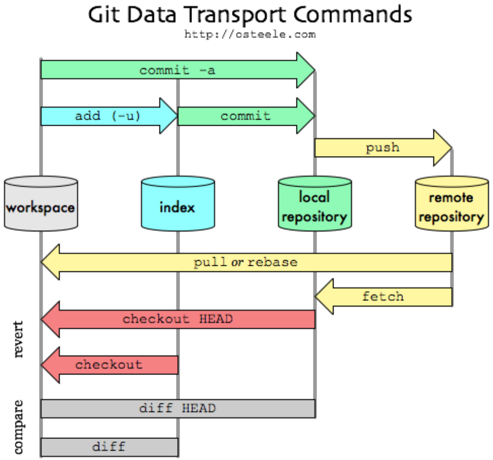
E-shop Demo

E-shop Demo

Git

Git

Git is a distributed version control system. We will use it during the course for the semestral work, as well as examples used in seminars. Git has several dozens of commands, but we need only a few of them.



Task 2 – Git

1. Go to `https://gitlab.fel.cvut.cz` and login using SSO (the `SSO login cvutID` button).
2. Go to the Projects screen.
3. In the list, you should see a project of the form of `B221_B6B36EAR/<CVUT-LOGIN>`, where `CVUT-LOGIN` is your CVUT login, e.g., `B221_B6B36EAR/ledvima1`. Click on the project name.
4. You will see a project containing one file - `.gitignore`.

Task 2 – (Git empty repository)

In case your repository is empty, here are your options.

Create a new repository

```
git clone git@gitlab.fel.cvut.cz:B221_B6B36EAR/{CVUT-LOGIN}.git
cd {CVUT-LOGIN}
git switch -c main
touch README.md
git add README.md
git commit -m "add README"
git push -u origin main
```

Push an existing folder

```
cd existing_folder
git init --initial-branch=main
git remote add origin git@gitlab.fel.cvut.cz:B221_B6B36EAR/{CVUT-LOGIN}.git
git add .
git commit -m "Initial commit"
git push -u origin main
```

Push an existing Git repository

```
cd existing_repo
git remote rename origin old-origin
git remote add origin git@gitlab.fel.cvut.cz:B221_B6B36EAR/{CVUT-LOGIN}.git
git push -u origin --all
git push -u origin --tags
```

Task 2 – Git (cont.)

Git SSH Keys

If you have not done so yet, we strongly recommend adding an SSH key to your Gitlab account. It greatly simplifies working with Git. Use the following tutorial: `https://gitlab.fel.cvut.cz/help/user/ssh.md#add-an-ssh-key-to-your-gitlab-account`

5. Clone your repository (*Clone with SSH*).
6. Create a `README.md` file and push it back into the remote repository.
7. Now you should also see your new `README.md` file in Gitlab.

We will call the local repository R_1.

Task 3 – Git Pull

1. Clone your repository once more as described in the previous section, but this time into a new local repository *test-pull*.
2. Inside the *test-pull* directory, edit the `README.md` file and change its content.
3. Perform the `add-commit-push` sequence for this file. (Note that `add` is needed for an existing file as well to schedule the *file change* for commit).
4. Run `git pull` inside R_1.
5. You should see the updated `README.md` file in R_1 now.

Task 4 – Git in IDE

1. Perform *Task 3* in your IDE.

Git Merge

Typically occurs when remote repository content changes during local repository editing. For instance:

1. Someone edits `README.md` and pushes changes into remote repository.
2. You edit `README.md` without knowing about the remote change.
3. You commit your changes.
4. You attempt to push but the push is rejected.
5. You need to merge the changes before attempting push again.

Git GUI clients **greatly** simplify this process.
Another possible strategy in this case is to use **rebase**.

Task 5 – Git Merge

1. Edit `README.md` in R_1 and do the regular `add-commit-push` sequence.
2. Without pulling changes, edit `README.md` in the *test-pull* repository you created in Task 4 in your favorite IDE.
3. Try to commit and push changes.
4. Push should be rejected.
5. Merge changes. IDE should let you edit the conflicts in `README.md`.
6. Commit the merge and push.

Note that if there is no conflict in file changes (i.e., the same file was not changed), Git clients are able to do the merge automatically and one just needs to push the merge.

The End

The End

Thank You

Resources

- https://cw.fel.cvut.cz/b221/courses/b6b36ear/start
- https://git-scm.com/docs
- http://blog.osteele.com/posts/2008/05/my-git-workflow
- https://docs.github.com/en/github/collaborating-with-pull-requests/working-with-forks/syncing-a-fork#syncing-a-fork-from-the-command-line