Seminar #1

Petr Křemen, Martin Ledvinka

1 Goal

This is an introductory seminar. You will become familiar with basic tools and setups used in the course.

2 Course Organization

2.1 Software Stack

The seminars will be based on the following software stack:

- ☑ Java 8
- ☑ Apache Tomcat 8
- ☑ Netbeans 8.2
- ☑ Maven 3
- ☑ NodeJS 6.5
- ☑ GIT 2
- ☑ PostgreSQL 9
- ☑ PgAdmin3
- ☑ Chrome
- ☐ Postman plugin for Chrome (not installed in the lab by default)

TASK 1: Installation Check

Lab computer should have preinstalled all the tools except the Postman plugin. Please install it manually.

Your own computer can be used as an alternative – make sure you have all the required software installed.

To test that your environment is ready, you should be able to run the test project. Download the project from
`https://cw.fel.cvut.cz/wiki/_media/courses/b6b33ear/setup-project.zip` and make it running, as described in `setup-project/README.md`.

2.2 Semestral Work

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.

Read carefully details about semestral work organization that is available at `https://cw.fel.cvut.cz/wiki/courses/b6b33ear/hodnoceni`.

3 GIT Basics

GIT [1] is a distributed version control system. We will use it during the course for semestral work, as well as examples used during seminars. GIT has several dozens of commands, but we touch only a few of them. The basic workflow is shown in Figure 1.

TASK 2: Obtain Your Semestral Work Repository

Let's learn GIT basics on the repository you will use for your semestral work.

1. Go to `https://gitlab.fel.cvut.cz` and login using SSO (the Shibboleth button).
2. Go to the *Projects* screen.
3. In the list, you should see a project of the form of `B161_<C-ID>/<CVUT-LOGIN>`, where `<C-ID>` is one of `B6B33EAR, BD6B33EAR, A7B39WPA, AD7B39WPA` and `CVUT-LOGIN` is your CVUT login, e.g. `B161_AD7B39WPA/kotlito1`. Click on the project name.
4. You will see a screen with informing you that you are inside an empty repository with a few commands that help you obtain the repository into your computer. Follow the sections *Git global setup* and *Create a new repository*. These steps clone your remote repository, create a README.md file and push it into the server back.
5. Now you should see in Gitlab your new `README.md` file.

We will call this local repository R_1.

[1] `https://git-scm.com`

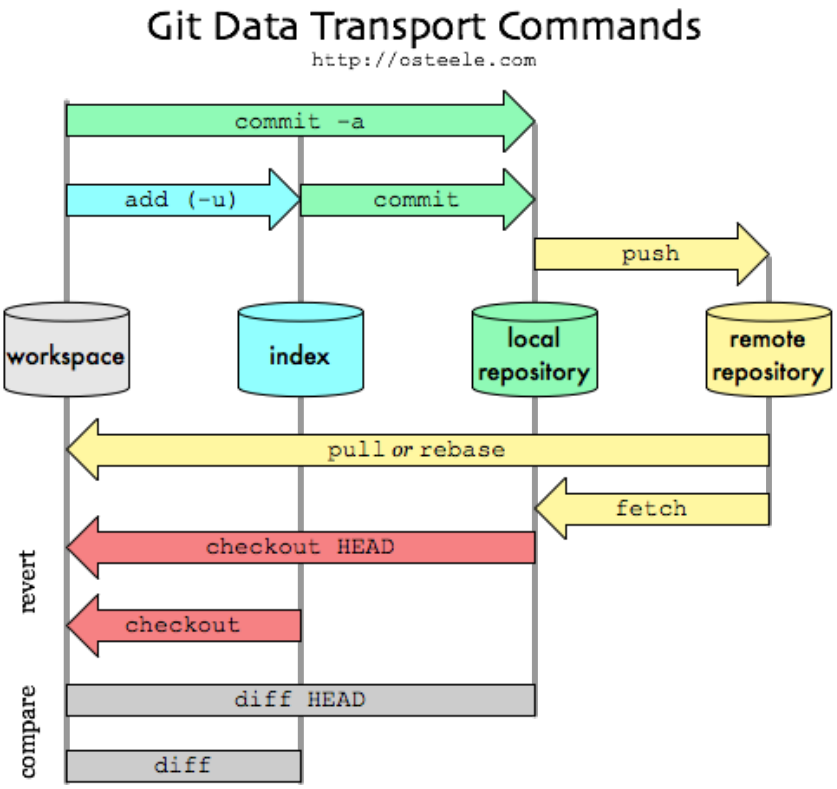

Figure 1: Basic GIT operations (taken from `http://blog.osteele.com/posts/2008/05/my-git-workflow`).

TASK 3: Pull Remote Changes

1. Clone your repository once more as described in the previous section, but this time into a new local repository `test-pull`:

   ```
   git clone git@gitlab.fel.cvut.cz:B161_<C-ID>/<CVUT-LOGIN>.git 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. Inside R_1 run

   ```
   git pull
   ```

5. Now you should have the updated README.md file in R_1.

Should you want to go deeper into GIT, refer to `https://git-scm.com/docs`, or many guides available on the internet.

Also, a comprehensive list of user interfaces for GIT is listed at `https://git-scm.com/downloads/guis`. Also, major IDEs like Netbeans, Eclipse, or Intellij have built-in or pluggable support for GIT, so you will mostly use their GIT support. Note that in order to propagate a change to the server you need to run **add**,)**commit** and **push** commands.

TASK 4: Work with GIT in Netbeans

1. Run Netbeans
2. Go to `Team` ▶ `GIT` ▶ `Clone` and enter the URL

   ```
   git@gitlab.fel.cvut.cz:ear/setup-project.git
   ```

   This will clone the setup project downloaded in TASK 1.
3. Select `Private/Public Key` and list your private key file (typically `~/.ssh/id_rsa`). Change the `Clone into` as needed. You can leave other options unchanged and proceed through the wizard.
4. Change the content of `Web Pages` ▶ `index.html` file. Now try to perform `add`—`commit`—`push` sequence for this file. You should succeed with `add` and `commit`, but fail for `push`. Explore the color highlighting of changes in the `Projects` tab of Netbeans.

4 Maven Basics

Maven [2] is a software project management tool used for controlling project's build, reporting and documentation. Maven is controlled by the configuration file `pom.xml`. Take a look into `pom.xml` for the project from TASK 4.

TASK 5: Install Maven artifact

The most important part is the identification of the project:

```
<groupId>cz.cvut.kbss</groupId>
<artifactId>ear-setup</artifactId>
<version>0.0.1</version>
<packaging>war</packaging>
```

[2] https://maven.apache.org

Group ID, artifact ID, as well as version compose together an identifier for the particular "project". You can compile, package and install the project into your local Maven repository by

```
mvn install
```

At this point, you should be able to see in `~/.m2/repository/cz/cvut/kbss/ear-setup/0.0.1` the WAR file of the project.

TASK 6: Dependencies

One of the most important Maven features is dependency management. In the `<dependencies>` tag you will find a list of projects that `ear-setup` depends on. For example, EclipseLink (a JPA implementation) is included on classpath of `ear-setup` by.

```
<dependency>
  <groupId>org.eclipse.persistence</groupId>
  <artifactId>eclipselink</artifactId>
  <version>2.6.2</version>
</dependency>
```

You can check that each dependency of `ear-setup` is also present in the local maven repository `~/.m2/repository/`.

1. Change EclipseLink version from 2.6.2 to 2.6.4
2. Run `mvn clean install`. As a result, Maven automatically fetches the library and installs it into your repository. Now, you should see the new EclipseLink in `~/.m2/repository/org/eclipse/persistence/eclipselink/2.6.4`.

TASK 7: Build Process

The build process can be controlled by build plugins. For example, the following `maven-war-plugin` configuration creates a `war` archive named `ear-setup`.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <version>2.4</version>
  <configuration>
    <failOnMissingWebXml>false</failOnMissingWebXml>
    <warName>ear-setup</warName>
  </configuration>
</plugin>
```

Insert the source plugin configuration into `project` ▶ `build` ▶ `plugins`:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-source-plugin</artifactId>
  <version>3.0.1</version>
  <executions>
    <execution>
      <id>attach-sources</id>
      <phase>verify</phase>
      <goals>
        <goal>jar-no-fork</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

and look what changes after `mvn clean install` execution in the `~/.m2/repository/cz/cvut/kbss/ear-setup/0.0.1` directory.