# Getting Started With GIT and Distributed Version Control

## Exercise - 1, Creating Remote Repositories)

The exercises below are team-exercises, so you only need one account per team, but I suggest that you all create a personal account. A central repository is perfect, also for individuals, as a backup media, and to synchronize work between different computers (laptop, desktop etc.)

If the only thing you need is version control (the unlimited undo list etc.) you don't need the central repository. Git can run 100% offline, and you only have to push if you actually want to share your work with someone.

## Exercise 2 - Class Demo)

This is a class demo between me and selected members from the class. Observe, since you will have to repeat most of what we do in teams after the demo.

## Exercise 3 -Team Exercise – Getting Started)

The following exercises are all meant for a team.

Pick one member from the team as **User1** and use the central repository created by him/her for the exercise.

### User1

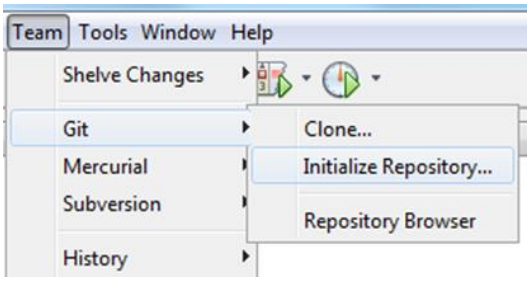
Here we start with a completely new **project** which will be set under version control

1) Create a repository on GitHub for this demo (use user-1's account)

2) Create a new plain java project

3) Clean and Build (To create ignore files/directories)

4) Choose initialize repository




5) Place the cursor on the project node (top) -> Select Commit

Provide a sufficient commit message

6) Select **Push** (See the section Pushing in the NetBeans Git Tutorial).

### User 2..n

Here you would like to get our own working copy of the newly published solution.

7) Select Team→ Git → Clone (See the section Cloning in the NetBeans Git Tutorial).

### All

8)

Start to work on the project (come up with something by yourself). First do it the right way. Communicate, communicate, communicate, and don't work on the same code (methods). Commit, Push and see how conflicts (if any) are solved automatically.

9) Now add some changes that will give a merge conflict and solve the conflict(s).

## Exercise 4 - Files that are hard/impossible to merge)

Sometimes we are using tools that generate files which we normally shouldn't touch. These files cannot be ignored since they are an important part of the code base.

Designing GUI code with NetBeans it a perfect example of this case (which it not the same as to say that NetBeans did a good job with this design ;-( ).

### User1

Here we start with a completely new **project** which will be set under version control

1) Create a new plain java project

2) Clean and Build (To create ignore files/directories)

3) Choose initialize repository

4) Add a JFrame Form to the project and drag some buttons into the Frame.

5) Place the cursor on the project node (top) -> Select Commit, and provide a sufficient commit message

### User 2..n

Here we would like to get our own working copy of the newly published solution

7) Team→ Git → Clone (See the section Cloning in the NetBeans Git Tutorial).

### All

8) Start to work on the project, add some more buttons, panels etc.

9) Commit and push, and see what happens unless you are the first one to commit.

### Task:

**10 ) Can these conflicts be (easily) solved? If yes do it, if not what is the lesson learned from this exercise.**

# Exercise 5 – A database project

## User1 (and only user1)

1) Follow step 1-6 from exercise 3.

2) Create a package (should have been a folder, but this will allow us to see it from the project view) called ***scripts*** in the "source packages" folder.

3) Right click the folder and select: New-> Other-> SQL file. Give it the name *setup.sql* and add the script below to the file:

```
DROP TABLE Person;

CREATE TABLE PERSON (
  ID INTEGER NOT NULL,
  FIRST_NAME VARCHAR(50),
  LAST_NAME VARCHAR(50),
  PRIMARY KEY (ID)
);

insert into Person values(100,'Peter','Hansen');
insert into Person values(110,'Lone','Hansen');
insert into Person values(120,'John','McDonald');
insert into Person values(130,'Peter','Jensen');
```

NetBeans provide an acceptable Database frontend, but for this exercise I will assume that you will use SqlDeveloper to create the table.

The main reason to put script into the project, is to get an easy way to commit it together with the java code that uses the table.

4) Open SqlDeveloper and execute the script to create the table

5)

It is a matter of debate whether you should put your database driver under version control. I will however recommend it for this semester, in order to avoid problems with the location of this file.

Add a new folder to your project called ***jars*** and copy the driver ojdbc7.jar into the folder.

Include the driver into your project as usual (you need to do this even though it's located inside your project).

The effect of this is that when you commit, you also commit the driver and also, your project has a reference to the driver which will work for all that pulls the project. That is, after a pull you are "ready to go".

6)

Create a new Java file called JdbcDemo and replace all content in the file with the code given below:

```
import java.sql.Statement;
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;

public class JdbcDemo {

  public static final String userName = "xxx";
  public static final String pw = "yyy";
  public static final String dbms = "oracle";
  static final String driverDerby = "org.apache.derby.jdbc.EmbeddedDriver";
  static final String driverOracle = "oracle.jdbc.driver.OracleDriver";
  static final String connectionOracle = "jdbc:oracle:thin:@datdb.cphbusiness.dk:1521:dat";
  static final String connectionDerby = "jdbc:derby://localhost:1527/myLocalDerbyDatabase";

  public Connection getConnection() throws SQLException {
    Connection conn = null;
    Properties connectionProps = new Properties();
    connectionProps.put("user", userName);
    connectionProps.put("password", pw);
    if (JdbcDemo.dbms.equals("oracle")) {
      conn = DriverManager.getConnection(connectionOracle, connectionProps);
    } else if (JdbcDemo.dbms.equals("derby")) {
      conn = DriverManager.getConnection(connectionDerby, connectionProps);
    }
    System.out.println("Connected to database");
    return conn;
  }

  public void personQuery() throws SQLException {
    try (Statement stmt = getConnection().createStatement()) {
      stmt.executeQuery("SELECT * FROM PERSON");
      ResultSet rs = stmt.getResultSet();
      while (rs.next()) {
        System.out.println("Id:" + rs.getInt(1) + ", " + rs.getString("FIRST_NAME") + ", "
                           + rs.getString("LAST_NAME"));
      }
    }
  }

  public static void main(String[] args) {
    JdbcDemo test = new JdbcDemo();
    try {
      test.personQuery();
    } catch (SQLException ex) {
      Logger.getLogger(JdbcDemo.class.getName()).log(Level.SEVERE, null, ex);
    }
  }
}
```

7) Change the username and password in order to connect to your own database. Compile, execute and verify that you can connect to the database and show the content of the Person table.

8) Write a commit message that clearly explains what you have done, and what must be done by others to execute the code (execute the script against their own development database).

REMEMBER the idea is, for each team member to use their own personal development database, so they should NOT use your username and password.

9) Commit and push

### User 2..n

10)

Clone the solution, and pull all changes.

Follow the instructions in the commit message.

Change password and username to your own and verify that you can execute the project

11)

The customer decides that a phone number is necessary (actually each person could have several phone numbers).

Let one member in the team implement this change (which involves changes to both the script and the code).

Write a commit message that clearly explains what have been done, and what must be done by others to execute the code.

Commit and push