Exercise 2: Git

Pre-Lab

Please have these completed before coming to the lab.

1. What are Source Code Management (SCM) systems? How are they useful?
2. Git is a SCM system. What is a repository in Git? How is it different from a directory?
3. Install Git on your computer. There are GUI clients for Git out there, but for this lab **you are not allowed to use them**. We'll be working exclusively with the command-line interface this semester.
4. Once Git is installed, you can configure your username and email address with these shell commands:
    a. git config --global user.name "FIRST_NAME LAST_NAME"
    b. git config --global user.email "MY_NAME@example.com"
5. You'll need a hosting service for your Git repositories. You have the choice between GitHub and GitLab.
    a. GitHub ist the most popular tool for managing Git repositories. It's free and if you use your HTW email address you'll get access to the Github Student Developer pack. It's likely that you'll want an account at some point in your future, so you might as well create one today.
    b. The HTW has a GitLab server for IMI students. You can only access it from the campus network, so you'll have to set up a VPN connection first. Again, you'll use the HTW VPN a number of times during your studies, so you'll might as well do the setup today.
    You can access the server with your HTW credentials, so you don't have to create an account if you're using GitLab.
6. One last step of setup remaining: Once you have access to a GitHub or GitLab account, you need to create a SHA key and add it to your account. Here are the instructions for GitHub and GitLab respectively.

Lab exercises

These are the required exercises for this week. Work in groups of two and turn in the same report for each member of the pair. Remember to put your names on the report, as well as the link to your groups repository. If you haven't already, I highly recommend switching to Eclipse now - using different IDEs makes collaborating with Git more complicated.

The goal for this week is that you are getting comfortable with the basic Git workflow. You'll be using Git for your pair programming this semester and include a link to your groups repository in every report. Git is a very powerful tool, but for this semester we'll stick to the 8 basic commands: *init, remote, clone, add, commit, status, push, pull*.

1. Create a remote repository for your group and a local repository for each group member. Add some files (a shopping list, some MS Paint art, a cheat sheet with the most important Git commands, a message for your partner, etc) to your local repository. Push everything to your remote repository. Get the files from your colleague (using Git pull) and make some changes to them. Push your additions back to the server, then get your teammates changes to see how they messed with your files.
2. Now practice the Git workflow with an actual Java project. You'll do the setup for your partner, then exchange your code and finish your partners exercise.
    a. Choose each an exercise from the medium logic puzzles that your partner hasn't solved yet. Create a project in your local repository and set up the empty method stubs required to solve the exercise, e.g.:

```
public int noTeenSum(int a, int b, int c) {
      throw new java.lang.UnsupportedOperationException("Not implemented
yet.");
}
public int fixTeen(int n){
      throw new java.lang.UnsupportedOperationException("Not implemented
yet.");
}
```

    b. Add comments to the method stating what it's supposed to do.
    c. (Optional) Create a JUnit test case for your class. Write assertions for each method, covering the edge cases.
    d. Save and push to your remote repository.
    e. Pull your partners' exercise and complete it.
    f. (Optional) Can you come up with edge cases not covered by your partners tests? Write them down in your report.

For the bored:

1. So far you've been working on different files the entire time. What happens if you both try to change the same file?
    a. Revisit a text file from the first part of this exercise. Modify a different line each, then push your changes.
    b. Now try to both modify the same line and push your changes. What do you expect to happen? What did happen?
        i. You can always return to the state of your last **local** commit using:
        git reset --hard HEAD
        You will lose all local changes since your last commit, so use with caution.

Your report is due by 10.00 pm the night before your next lab! Your report should include any collaborators, summarize what you learned, and note the time you invested in this exercise. What operating systems were you using? Did you run into any OS related problems?