Lab 1 – Version Control and IDE Setup

Introduction

You will work on this in pairs. The instructions specify who should do what, written as Person A and Person B. If there are three of you in your group, then one person should be A and the others can be B.

Sign up for a cloud-based Git provider

Everyone in your group must have an account with the same cloud-based Git provider. We suggest GitHub.

1. If you do not already have an account with the chosen cloud-based Git provider, sign up now.

Prepare a new cloud-based git repository

Person A (only) should follow these steps:

1. Within the web portal of your cloud-based Git provider, create a new repository named **labs** (or something similar).
2. Note the URL of the new repository. For example, it might be **https://github.com/fredbloggs/labs**
3. We'll refer to this URL later in these instructions as **<new-repository-url>**

Populate the new repository

Person A (only) should follow these steps:

1. Clone the GitHub repository **https://github.com/qa-apprenticeships/SDL4M5-labs** into a local folder on your computer at **c:\labs**, using:

   git clone https://github.com/qa-apprenticeships/SDL4M5-labs.git c:\labs
   you will now have a copy of all the files.

2. Within your **c:\labs** folder, delete the **.git** folder (you may need to enable the display of hidden files first).
   We just wish to copy the files at this stage.

3. Prepare to store this folder in your new repository, using these instructions

   cd c:\labs

   git init

   initialises a repository inside the folder

   git add .

   All the files will be staged

   git commit -m "Initial version"

   stores a snapshot of the files in the repository at a certain point in time

   You can have multiple commits and rewind to an earlier version when necessary

   git branch -M shared
   This will create a pointer to a snapshot of your changes, so you add a new feature or make small or large changes to your code.

   git remote add origin <new-repository-url>
   push your changes to enable collaboration with others.

   origin is the shorthand name for the repository where was originally cloned.
   remote is a common repository that all team members use to exchange their changes.

   git push -u origin shared
   push the change to the Owner's repository on Git

   Once you have made the changes, you can push them to the remote repository. Your code will now be safely held in the cloud!

Invite other team members as collaborators

1. Invite the other member(s) of your team to be collaborators in the repository (this is the only way they will have permission to push changes into the repository).
   - In GitHub, you can do this on **Settings** / **Manage access** / **Invite a collaborator**
2. The other team member(s) should receive an invite via email, telling them how they can accept the invitation to become a collaborator on the repository. Accept the invitation, so you have permission to access and push to the repository.

Clone the new repository onto the machines of other team members

These steps are just for the other team members – not for person A.

1. Clone the new repository to your computer, using:

   git clone <new-repository-url>.git c:\labs

Experiment with the shared branch

The shared branch of the repository is for exercises where all members of the team will be working on the same copy of the code, and changes made by one member should be seen by the other members. For example, we will use this approach during the 'pair programming' exercises.

Coordinate the following steps between yourselves, so that everyone gets a go at each step...

1. Look in the "**Other"** folder - you should see three files:
   file-1.txt, **file-2.txt** and **file-3.txt**
2. One person should edit one of the files, save it, then push it up into the cloud repository, using:

   git add .

   git commit -m "my comment"

   git push

3. The others should pull the changes from the cloud repository into their local repository (see below), then check the contents of the files - you should see the changes that the other person made

   git pull

4. Take it in turns, changing a file, pushing it, and having the others pull it back down.
5. Try setting up a scenario where two of you both edit the same file at the same time. Both try pushing your changes. If Git was not able to automatically merge your changes together, you will be notified, and will need to manually merge the changes together.

6. To do that, first pull down the latest changes

   git pull

7. You will be told which files could not be automatically merged. Edit that file, and you should see some extra markings
   <<<<< HEAD ====== etc., around the area that needs merging. Make whatever manual

changes you think are necessary to keep both your change and the other person's change. Then, push the file back in the normal way.

git add .

git commit -m "my comment"

git push

8. The other team members should then refresh their files too, so everyone is synchronised again.

   git pull

Experiment with individual branches

Sometimes, we may be doing exercises where all members of the team will need to work on their own individual branch of the code, so that changes made by one member do not overwrite the changes made by another member, even when everyone synchronises their shared branch. In these cases, we'll use a separate 'named individual' branch.

Everyone in the team should do the following:

1. Create a new branch in your repository, using your first name as the name of the branch (assuming all members of your team have different names - choose something else if not!)

   git branch <your-name>

2. Next, switch to your new branch

   git checkout <your-name>

3. Everyone change the same file (e.g., file-1.txt), then push it back to your own branch.

   git add .

   git commit -m "my comment"

   git push

4. NB: The first time you do this, you'll need to specify the remote branch that your new branch maps to:

   git push -u origin <your-name>

5. Check that doing a pull doesn't bring down other people's changes (they're all on different branches to you):

   git pull

6. Switch back to the shared branch:

   git checkout shared

7. Check the contents of the files - they should have reverted to how they were when you were all sharing them.
8. Switch back again to your individual branch:

   git checkout <your-name>
9. Check the contents of the files - they should again contain your individual work.
10. Finally, switch back to the shared branch:

    git checkout shared

Prepare your IDE

We will use various languages for labs, so you may need several different IDEs available. Some labs only come in one language; others come in several. You can choose which language to work in, and can switch around as we proceed, depending on your preferences. For example, if you finish a lab in one language, you might like to try repeating the lab in another language – in particular where the IDE tools or library packages are different.

In the following sections, we give specific instructions for each IDE / language.

Java/Eclipse

1. Make sure you have Java SKD 1.8 (Java Version 8) installed.
2. Open Eclipse.
3. Make sure that the Java SDK 1.8 (JavaSE-1.8) is registered as an available Java Runtime (JRE).
4. Open a new workspace, which is c:\labs\java
5. Import all the Java projects within c:\labs\java into the workspace

C#/Visual Studio

To work on a lab in C#, simply open the appropriate .sln file by double-clicking on it.