Programmeertheorie
Git

Why?

If you have been following the minor program, then so far you have not done any programming in a team before. It is not hard to imagine that programming in a team is somewhat difficult, because missing just one character can cause your program to crash or not even compile. Hence, working in for instance Google docs or Dropbox is not an option. Just imagine the file you are working in changing all of a sudden because some teammate decided to modify some code.

At this point you might think of having every person in your group work in a different file, and then at the end you merge it together. However, this is very time consuming and difficult. For what if you have this brilliant idea, but you have to modify something your teammate is or was working on, and you cannot reach your teammate at three o' clock in the morning after a whole night of coding? Then frustratingly you have two options, either go to bed, or break the rules and change the code. Logically, the first option does not even cross your mind, and you change your teammate's code. Just so happens, your teammate had not shared his latest version. Now when you try to merge your code with his/hers the next day, you either have to redo your work or your teammate's. Sounds far fetched? It happens more often than you think (except the late hours hopefully).

Version control systems can partially remove the need for manual copy-pasting between versions, and that is why we introduce *Git*. Git is such a version control system and currently a quite popular one. Git allows you to keep versions of your code, documents or anything really. But more importantly, it allows your team to work in parallel. For Git can automatically *merge* versions by looking at the changes made from previous versions.

So for instance, in the early morning you meet with your group and you start coding. You implement the function you were going to build within no time. You tested it, and it works perfectly with the existing code. Now you can tell Git you have got a new version of the code, by *committing* your changes and finally *pushing* your version to the server. Two new words, but you will get it once we start practicing.

Obviously, you have time to spare and can either pity your slow teammates or grab some coffee. During this activity one of your teammates finishes whatever he was building, and commits his changes. This is where the Git magic kicks in, and Git will tell your teammate that he cannot do this, for he does not have the newest version, the one you committed. Now, your teammate has to *pull* your version from the server. If you and your teammate did not modify the exact same line in two different ways, Git will *automatically merge* your version with your teammate's version automatically. Your teammate can then test if the code still compiles and works. If so, your teammate can commit his changes and push them to the server. Cool huh?

If your last teammate tries to do the same, but did actually change the same line of code you did, then Git cannot automatically merge the two versions. Instead Git will tell your teammate there is a merge conflict, which your teammate can resolve by going into the file where the conflict remains. In this file Git will replace the line you have both modified by both versions of that line. It is up to your teammate to choose one or manually merge both lines. After this is done, your teammate can mark the conflict as resolved and can then commit and push.

Ultimately, Git solves the need for manually copy pasting code between different versions Git also allows you to *roll back* to any version within the repository. So if your planned new feature does not work out the way you wanted, you can always *revert* the changes.

What?

Git takes some getting used to. Therefore, we ask you to first of all install Git, and then practice a bit in an online tutorial.

How?

Register at GitHub

First, head on over to `https://github.com/` and register. Remember that server mentioned in the why section which was used to collaborate with your team members? That is the role GitHub is going to fulfill. Github is a company that can host Git repositories for you. On top of that, GitHub is "free" to use. However, GitHub has two types of repositories, private and public. As a non paying user, you can only create public repositories and here public means public. That is, anything you store in that repository can be found and downloaded by anyone on the web. Luckily GitHub likes students and you can apply for a student account at `https://education.github.com/`. With such an account you can have five self-created private repositories.

Create a repository

When you have registered at Github, have one of your group create a repository at `https://github.com/new`. You can choose whether it is public or private, and do not worry, you can always change it. Then, add your teammates as collaborators to that repository by going over to https://github.com/YOUR_USER_NAME/YOUR_REPO_NAME/settings/collaboration. Once that is done, add the staff as well:

- Jelleas - Jelle van Assema
- ...
- ...

Installing Git

So now you are registered, and your group has a repository on GitHub. Time to install.

CS50 Appliance

If you have not been following the minor program, you have probably never heard of the CS50 Appliance and that is no problem. Simply scroll down to your operating system and follow the instructions there. If you have been following the minor program, you can choose to continue your work in the CS50 Appliance. Git comes pre-installed on the Appliance, but we suggest you update to the newest version by simply typing the following command in your terminal:

```
sudo yum install git
```

Now, tell Git your name so your commits will be properly labeled:

```
git config --global user.name YOUR_NAME
```

And finally, tell Git the email address that will be associated with your Git commits. The email you specify should be the same one you used to sign up for GitHub.

```
git config --global user.email YOUR_EMAIL_ADDRESS
```

To get the repository you opened on GitHub on your computer simply enter the following:

```
git clone https://github.com/YOUR_USER_NAME/YOUR_REPO_NAME
```

To prevent typing your password every time you commit or push to Git you can cache your password. The following will have Git cache your password for an hour.

```
git config --global credential.helper ’cache --timeout=3600’
```

Windows and Mac (OS X)

To install Git on Windows or Mac we suggest downloading the GUI build by GitHub at `https://mac.github.com/` and `https://windows.github.com/` respectively. Both come with a command line interface so you can still enter text based commands. And sadly, you will most likely have to at some point in time. However, most of the time the GUI will offer you all the tools you need.

Linux

Follow the instructions at `https://help.github.com/articles/set-up-git/#platform-linux`. As you might notice, the instructions are quite similar to those for the CS50 Appliance. That is because the Appliance is simply a version of Fedora.

Practice

To get you somewhat accustomed to Git we recommend you follow the tutorial at `http://pcottle.github.io/learnGitBranching/`. You should at least do the first two rows of the main levels and the first row of the remote level. It is a lot to take in, but do not panic just yet. Git is something that grows on you. Similarly to programming, it just takes time and effort, and probably some frustration from time to time.

When?

You should have opened and shared your team's repository with the staff before the first meeting.