Usage of Git Version Control

April 28th, 2020

1 Introduction

Git is a version control system and has been developed to allow many developers to work on the same project at the same time. Therefore, it tracks every change that has been done to a project per line instead of per file. Additionally to just having your files backed up in remote storage, which could be achieved with typical shared folders (Dropbox, Google Drive, etc.), Git has several advantages:

- Concurrent modifications to the same file are possible. When one developer edits a few lines in a file, and another developer edits different lines, Git can merge both changes without user interaction. Only if both developers edit the same lines at the same time, the user might be asked to clear conflicting changes.
- Every past state that has been checked into Git is recoverable easily. Imagine, for example, you notice that the changes you did yesterday broke something that has worked before. With Git, you can see exactly what you have changed, or set the state of your project back to the working state.
- You can do all this offline and synchronize the changes with the server at a later point in time.

However, due to the per-line change management, Git is only useful for text-based files (source code, xml, latex and similar). It is not useful for binary files (Microsoft Office documents, video or audio files).

This tutorial will introduce you to the basic concepts and usage patterns of Git. We will assume Git is already installed and we will be using Visual Studio Code (VSCode) during this tutorial. Therefore, **you should have completed the Toolchain Tutorial (Stud.IP) before**.

During the tutorial, you connect to your group's repository, make some changes, and upload those changes to the Gitlab server. You will also learn, how to upload changes from the Gitlab server to your repository. Lastly, we will have a look on the most common problems and how to resolve them. If you prefer, we also prepared a **video tutorial** that shows how to go through the whole process described in this tutorial. It can also be found on Stud.IP.

2 Clone the initial repository

For the exercise, every group gets their own repository. This is located at a Gitlab server of the university. You will be notified via email as soon as your repository was created.

To see your Git repository in the web-view, navigate with the browser to https://collaborating.tuhh.de. Log in with your TUHH credentials (that you also use for Stud.IP). If you have already been assigned to a group repository, you should see the project in the list after signing in. Select the project. There will be an overview of the project, where you can click on a blue *Clone* button. Copy the link unter the option *Clone with HTTPS*.

Start VSCode and open a new, empty folder. To do so, click on *File* → *Open Folder*. Navigate to a path, where you want to store the project folder and open it. Now, in the top menu bar, click on *Terminal* → *New Terminal*. A new terminal window will open.

In the terminal, enter the following lines and replace the tags in <> brackets accordingly. If Git clone asks you for your credentials, enter the TUHH credentials again.

```

    git config --global user.name "<Your Name>"
    git config --global user.email "<Your Email>"
    git clone <your-clone-link>
```

Now you have cloned your repository onto your computer. It contains a workspace definition, that you have to open with VSCode. Click on *File* → *Open Workspace* and select the file *ses-workspace.code-workspace* in the root directory of your repository.

Now you are connected to Git and ready to code. In the next section, we will make some changes to source files and synchronize them with the server.

3 Commit and Push changes

In Git, every change is called a **commit**. We will now learn how to make a commit. Before we can do that, we have to make some changes to the project files. Go to the source file in the first exercise folder at *task_1-1/src/main.c* and play around. For example, change the blink frequency from the LED by modifying the delay time.

On the left side of the VSCode window, click on the *Source Control* tab. You see a list of files, that have changed since the last commit. For now, that should only be the *main.c* file. If you click on it, you can see exactly what has changed in the file from the previous state. To commit the changes to the repository, select the *main.c* file in the list, right-click on it and click *Stage Changes*.

Once these changes are staged, they are marked for a commit. On the top of the *Source Control* view is a text field, where you have to enter a commit message. Commit messages should be descriptive, so that later on it is easy to understand why you have made a certain change, e.g., *Changed blink frequency*. Then click on the little checkmark over the field to commit.

The commit you have made exists now only locally on your machine and is not yet uploaded to the server. Hence, other collaborators, such as your group partner, will not see it. The operation to upload your commits to the server is called a **Push**. Next to the checkmark that you just pressed, is a button with the ··· symbol. When you click on it, a menu with several Git commands pops up. Click *push* to upload your commit to the server.

4 Pull changes

Remember that you are sharing a repository with your partner. Hence, when your partner uploads commits to the repository, you should also sync them with your computer. The operation to load commits from the server is called **Pull**.

Your local repository and the repository on the server are not automatically synchronized, but you have to pull changes from the server deliberately. To do so, click again on the ··· button and select *Pull*. Note, however, that you can not pull, while you have unstaged modifications. Therefore you need to get rid of them somehow. You can either commit them, if they are in a commitable state, or, to just temporarily hide your changes, you can use the *Stash* function.

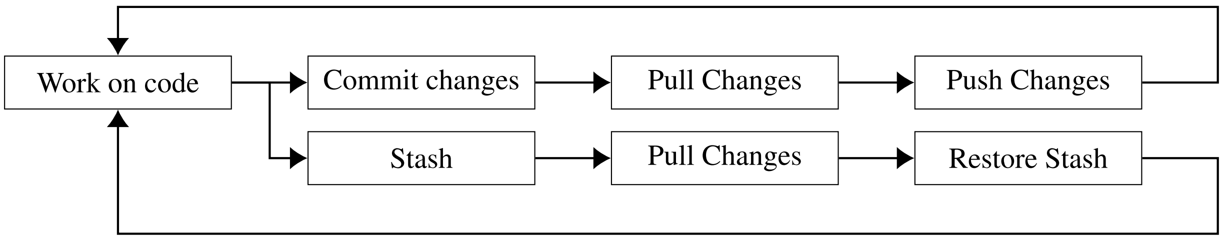
4.1 Stash

When you stash changes, they are stored in the background while your local repository appears to be in the state it was in after the last commit. You can stash changes via $\cdots \rightarrow$ *Stash*. In the same menu, you can press *Apply Latest Stash* to restore the previously stashed changes.

5 Conflicts

When two or more users edit the same lines in a file, Git can not know which change it should take. This is called a conflict and will require user input to be resolved. So when you pull changes from the server and you get a message saying that conflicts appeared, you can do the following: Go to the *Source Control* view, now you should see the conflicting file marked red within the *Changes* section. Click on the conflicted file. The upcoming view highlights all conflicting parts of the file, and both versions are shown. Your task is now to decide manually which changes to take and which to reject. Once you have done that, right-click on the file in the *Source Control* view and stage it. A commit message is automatically generated saying that you merged the conflicting changes in the files. You just need to click on *commit* now.

6 Complete Workflow



6.1 General Advices

- **Commit Frequency:** In general, you should rather make many small commits instead of a few big ones. Whenever you add a new feature or fix a bug, commit the change. Make sure, however, that the commits do not break working code.
- **Use descriptive commit messages:** Your commit messages should describe the purpose of your changes, not the changes itself. For example, "*Changed file main.c*" is not a descriptive commit message. A better message could be "*Fixed occasional crashes when button is pressed*", or "*Implement Temperature Sensor Driver*". A big plus of descriptive commit messages is, that you can always see later on what you actually wrote a line of code for in the first place.

In this tutorial, everything you need to know about Git for this exercise is described. However, there are many more useful features of Git. If you are interested to learn more about them, you can read about them in detail in the *Git Pro Book*, which is available at https://git-scm.com/book/en/v2.

A handy overview about the important Git commands is given at https://github.github.com/training-kit/downloads/de/github-git-cheat-sheet/.