Working with remote repositories and branches

In this section we will learn how to clone a repository from GitLab, how to find information about that repository, how to push information back to the repository, and finally how to work with multiple branches.

Relevant Links

- Remotes[1]
- Branches[2]

Notes

First of all, you should have already forked the WebAppsLabs[3] project over to your account. Make sure you have that open in your browser.

Cloning

The first thing we need to do is "clone" this project on your computer. To do that, find the "clone URL" section on the middle of your project webpage, and click on "Copy to clipboard" next to it. Now open up the Terminal, and navigate to the location where you want to store this project. The Desktop or some sort of Documents folders would be good places. The cloning process will create a folder itself, so you do not need to create the project's folder, you just need to be in the location where you want it created. Then you would type:

```
git clone pasteTheLinkHere
```

The link should look something like: "https://gitlab.com/yourNameHere/WebAppsLabs.git".

If you do a `ls` listing in the directory you should now see this new folder, called WebAppsLabs. Switch to it with `cd WebAppsLabs`. You should find a `README.md` file in there.

Learning about Remotes

The repository that GitLab has of your code is called the "remote". You also just cloned a full version of it locally on your computer. We will see how to interact between the two.

Type the following to see the remotes you have:

```
git remote –v
```

[1] http://git-scm.com/book/en/v2/Git-Basics-Working-with-Remotes
[2] http://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell
[3] https://gitlab.com/skiadas/WebAppsLabs

You should see “two” remotes, one for **fetch** and one for **push**. Fetch is what happens when we download changes from GitLab to our computer, Push is the opposite.

If you have collaborators, you can create extra “remotes” linking to their versions of the project, so that you can fetch changes they make. In our case, we will create such a remote that would access my main version of the project, so that you can get updates when I add new labs (your forked project won’t update directly). Do the following:

```
git remote add mainSource https://gitlab.com/skiadas/WebAppsLabs.git
```

Now if you look for all the remotes you should be seeing two new remotes (well really one remote with fetch and push). We will use this much later.

Adding remotes is not something you have to do often, usually only when you set up your project.

Fetching data

When you want to get updates from the remote, you use fetch:

```
git fetch origin
```

Probably not much will happen, since we just cloned the project, so we already have the newest state. Oftentimes you can skip the “origin” part and just say `git fetch`. We will also often use a variant, `git pull`. You can read about the difference in the book. With the workflow we will follow, there won’t be a difference.

When you want to submit updates, you would do (don’t need to do it now):

```
git push origin master
```

Or something of the kind, we will discuss “master” later. A simple `git push` often works as well.

You can find info about a remote with (try now):

```
git remote show origin
```

What you should notice there is that there are (at least) two remote branches tracked, Lab1 and master.

Branches

Branches are “paths of development” of your code. Each branch has a base commit that it springs from, and commits you make become part of this new branch, rather than the “main” branch you were on. But you can also merge one branch into another, which can complicate matters a bit. We won’t worry about that for a while.

For now we will just learn how to “switch” to a new branch. The command for that is “checkout”. In this case, we will check out the Lab1 branch:

```
git checkout Lab1
```

This might seem like an innocent little sentence, but in fact it completely changed what is in your working directory right now. To see this, do an `ls` listing. You should see a bunch of files there. Go back and forth between the two "branches" and look at each time what your directory looks like:

```
ls
git checkout master
ls
git checkout Lab1
ls
```

Take a moment to ponder this! The same directory in your computer can look completely different depending on which branch or which commit you are looking at. Later on in class we will do a checkout of Lab2, Lab3 etc. It will always be the same "directory", but it looks very different each time.

Before starting any new work, you need to always make sure you are in the correct branch.