LAB #3 - GIT (Supplementary)

Overview

Step 1 : learn to use `git` from within Visual Studio Code (VSCode).
Step 2 : learn more about `git` commands.

✓ Also see this example how to use VSCode and `git`. You can find a video here.

Margin notes contain ! and ✓ symbols. ! indicates that the line/paragraph is of uttermost importance and ✓ indicates that the line/paragraph contains something interesting or useful but not strictly mandatory.

Step 1 – Visual Studio Code and `git`

! Install extension such as `GitLense` and `Git Graph`.

✓ Search for commands by pressing `CTRL+SHIFT+P` and type the command.

Clone your repository by pressing `CTRL+SHIFT+P` and typing `git clone` (see Figure 1).

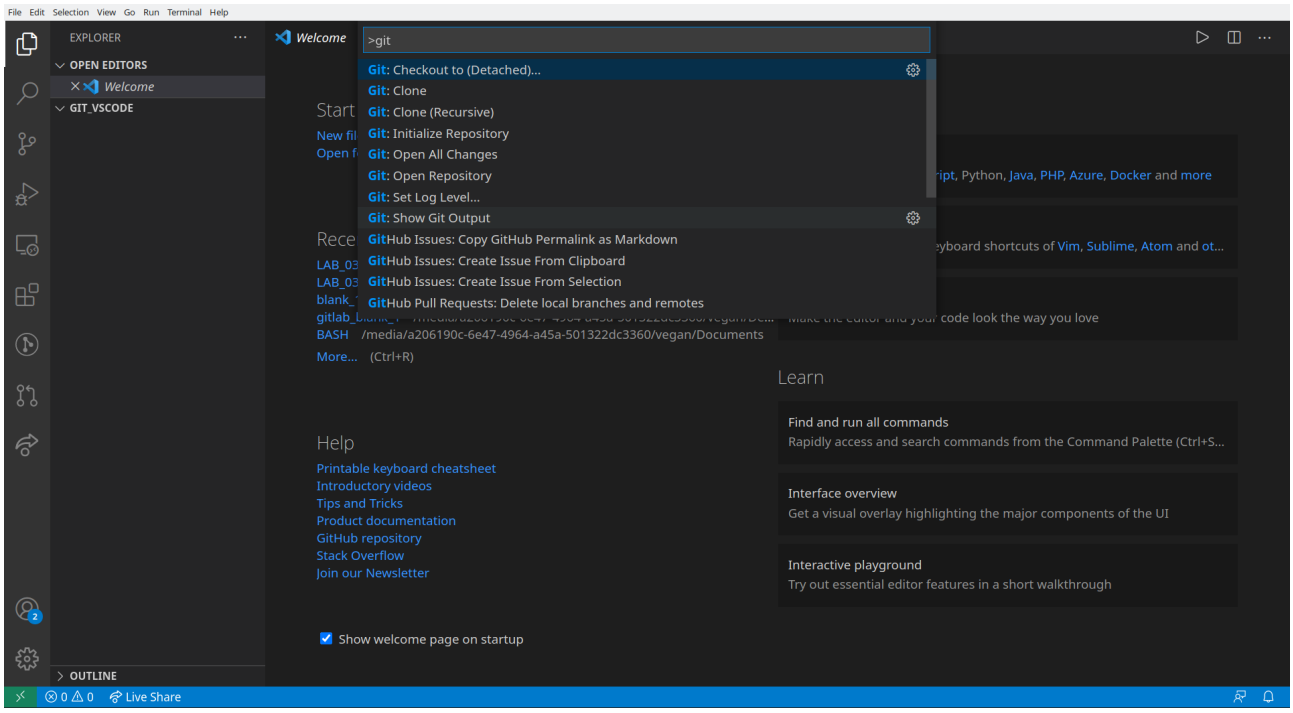

Figure 1: Search commands

Enter the address of your repository to start cloning (see Figure 2).

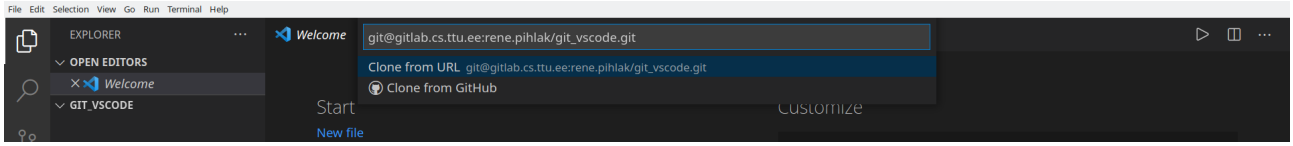

Figure 2: Clone a repository

Add files, such as `.gitignore` and add them (stage them) to be committed to the remote repository.

The `.gitignore` file allows you to blacklist files and folders to prevent that `git` uploads large or secret files and folders to the remote repository (see Figure 3).

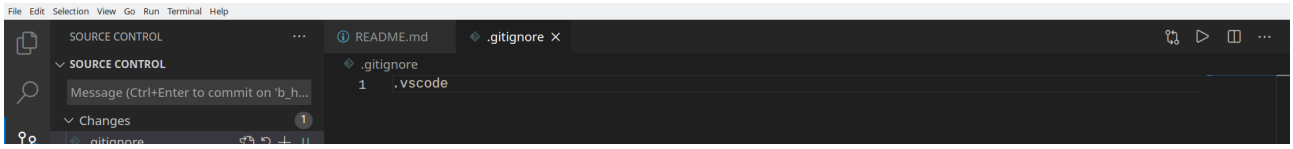

Figure 3: Add `.gitignore` rules

In VSCode, '`git stage all changes`' refers to a command '`git add -A`' (see Figure 4).

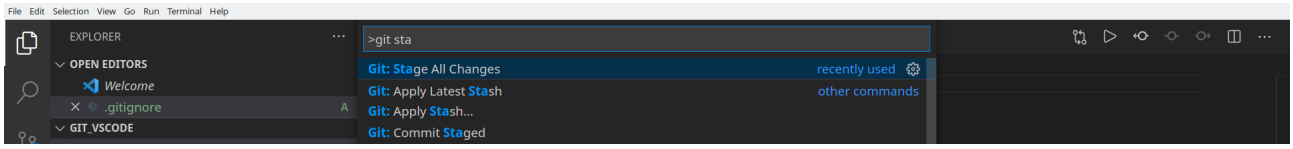

Figure 4: Add (stage) changes

Commit the changes (see Figure 5).

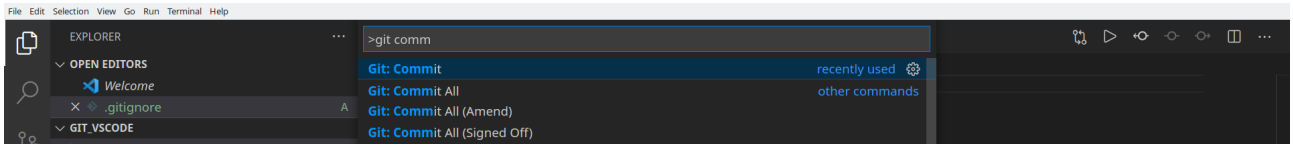

Figure 5: Commit changes

Push to remote repository (see Figure 6).

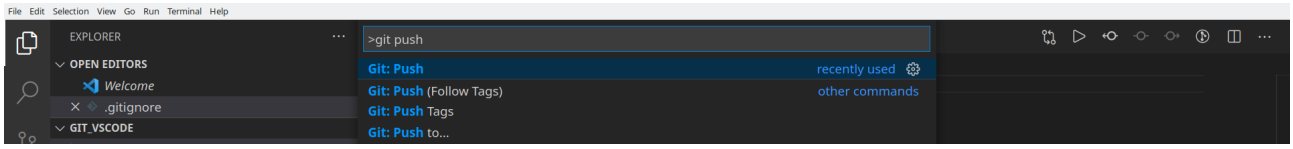

Figure 6: Push to remote repository

Check in `gitlab` that your changes appear there (see Figure 5).

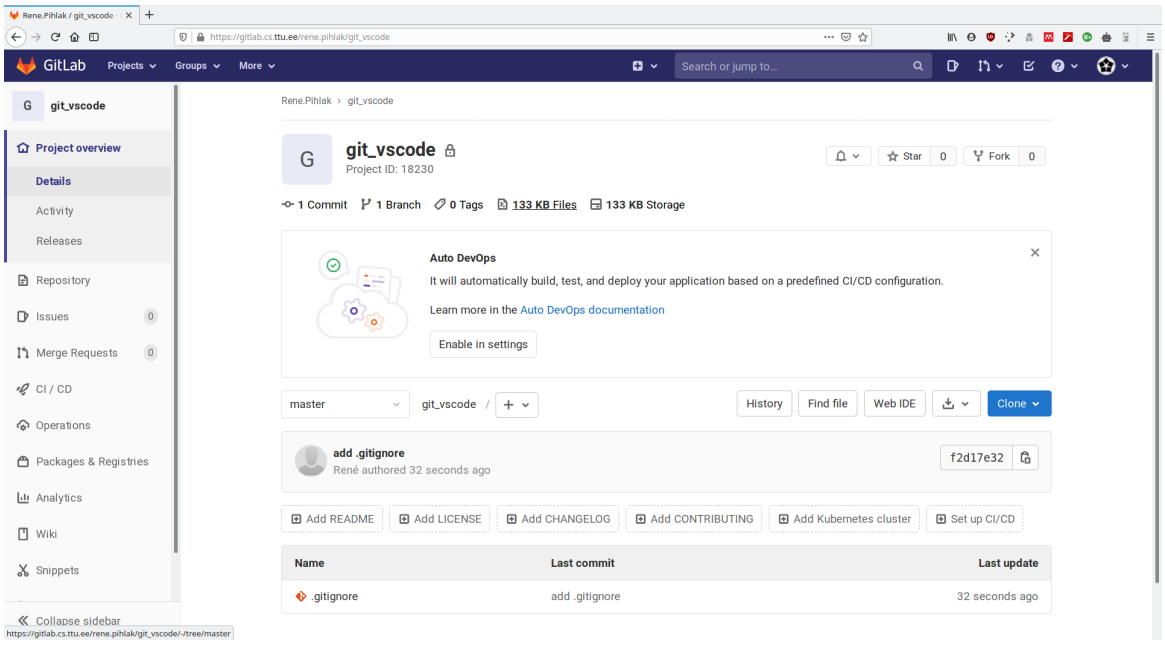

Figure 7: Check your changes in gitlab

Create new branches (see Figure 8).

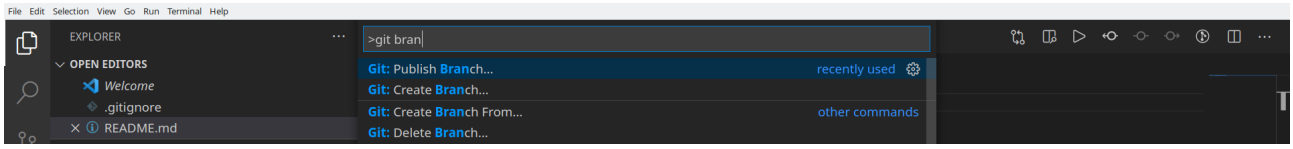

Figure 8: Create a new branches

Create new branches (see Figure 9).

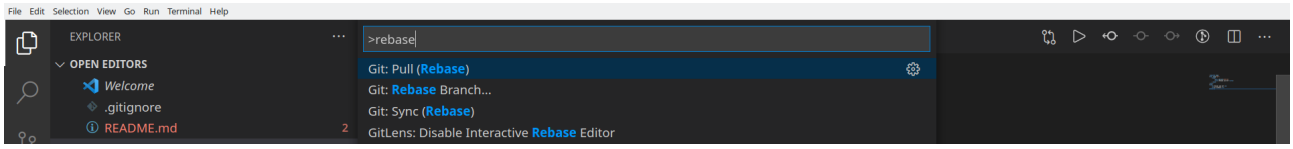

Figure 9: Rebase a branch

Resolve conflicts (see Figure 10).

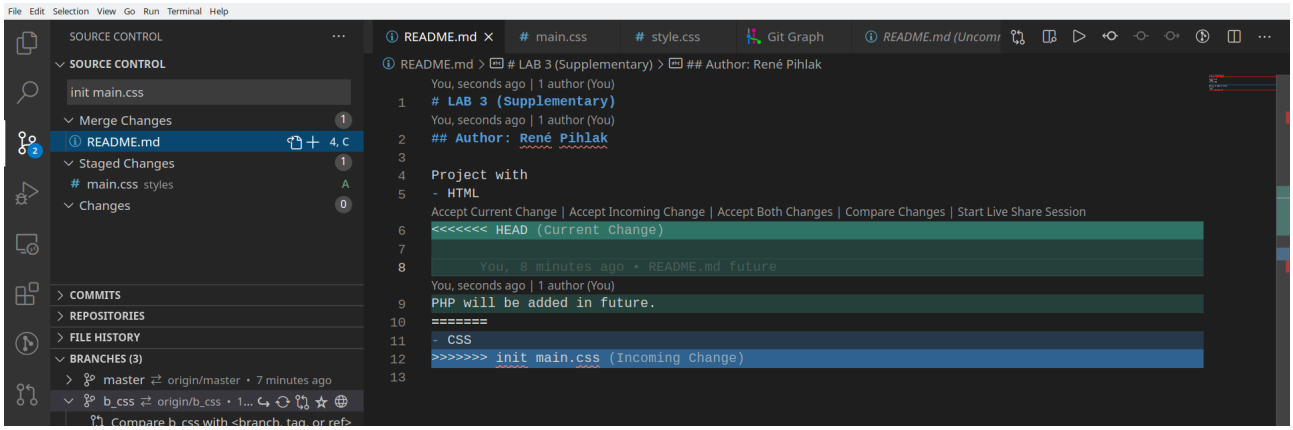

Figure 10: Resolve conflicts

Visualize your repository with a graph diagram (requires `Git Graph`) (see Figure 11).

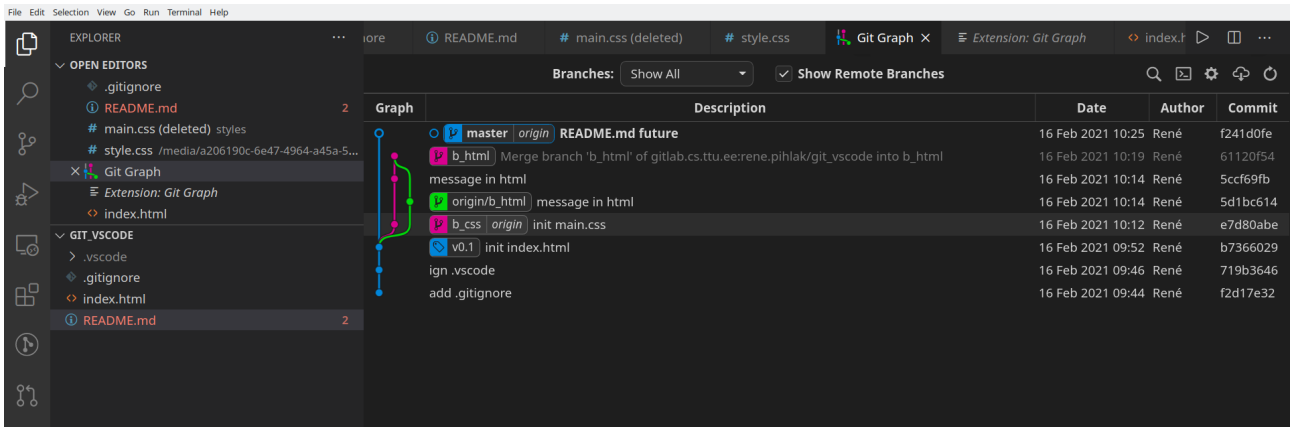

Figure 11: Graph diagram

Remove unnecessary branches and use `terminal` for `git` commands (see Figure 12).

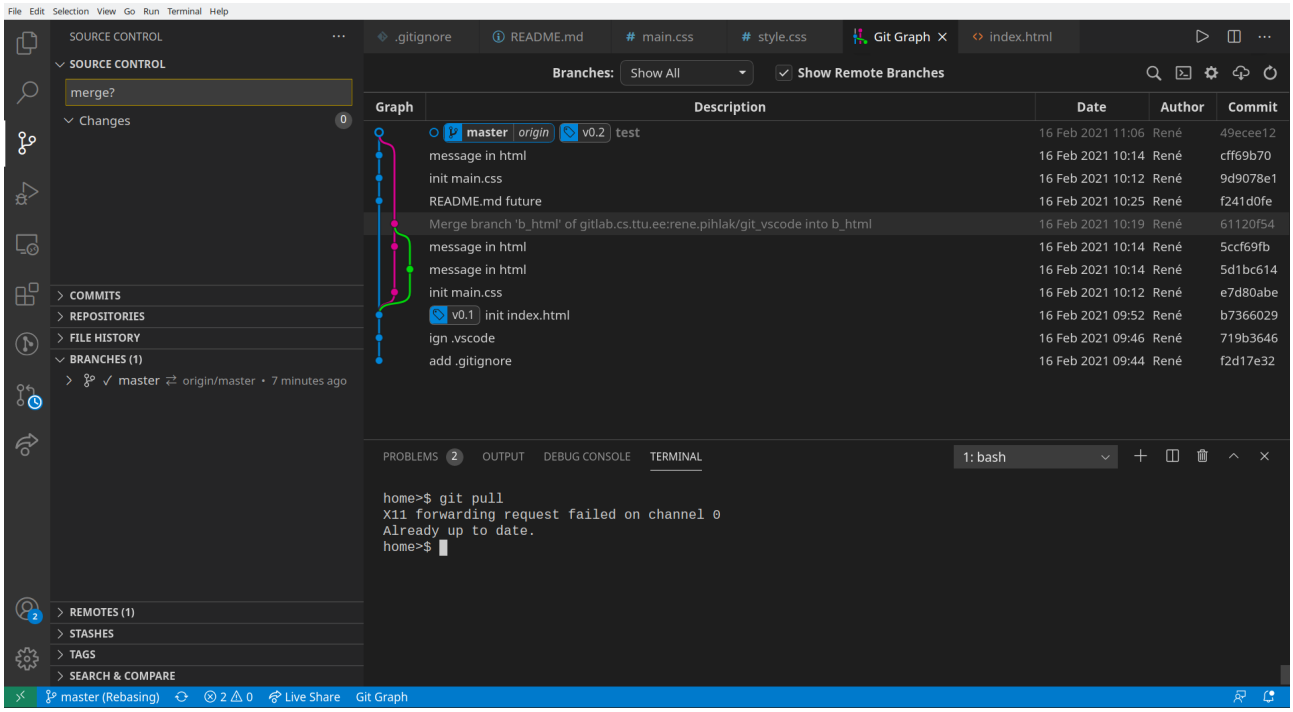

Figure 12: Use terminal and visualize repository without excess branches

Step 2 – Overview of `git` commands

Git task	Notes	Git commands
Tell Git who you are	Configure the author name and email address to be used with your commits.Note that Git strips some characters (for example trailing periods) from user.name.	`git config --global user.name "Sam Smith"` `git config --global user.email sam@example.com`
Create a new local repository		`git init`
Check out a repository	Create a working copy of a local repository:	`git clone /path/to/repository`
	For a remote server, use:	`git clone username@host:/path/to/repository`
Add files	Add one or more files to staging (index):	`git add <filename>` `git add *`
Commit	Commit changes to head (but not yet to the remote repository):	`git commit -m "Commit message"`
	Commit any files you've added with git add, and also commit any files you've changed since then:	`git commit -a`
Push	Send changes to the master branch of your remote repository:	`git push origin master`
Status	List the files you've changed and those you still need to add or commit:	`git status`
Connect to a remote repository	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	`git remote add origin <server>`
	List all currently configured remote repositories:	`git remote -v`

Git task	Notes	Git commands
Branches	Create a new branch and switch to it:	`git checkout -b <branchname>`
	Switch from one branch to another:	`git checkout <branchname>`
	List all the branches in your repo, and also tell you what branch you're currently in:	`git branch`
	Delete the feature branch:	`git branch -d <branchname>`
	Push the branch to your remote repository, so others can use it:	`git push origin <branchname>`
	Push all branches to your remote repository:	`git push --all origin`
	Delete a branch on your remote repository:	`git push origin :<branchname>`
Update from the remote repository	Fetch and merge changes on the remote server to your working directory:	`git pull`
	To merge a different branch into your active branch:	`git merge <branchname>`

	View all the merge conflicts. View the conflicts against the base file. Preview changes, before merging.	`git diff`
		`git diff --base <filename>`
		`git diff <sourcebranch> <targetbranch>`

	After you have manually resolved any conflicts, you mark the changed file:	`git add <filename>`
Tags	You can use tagging to mark a significant changeset, such as a release:	`git tag 1.0.0 <commitID>`
	CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:	`git log`
	Push all tags to remote repository:	`git push --tags origin`
Undo local changes	If you mess up, you can replace the changes in your working tree with the last content in head: Changes already added to the index, as well as new files, will be kept.	`git checkout -- <filename>`
	Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:	`git fetch origin  git reset --hard origin/master`
Search	Search the working directory for foo():	`git grep "foo()"`