**PEG Setup Document**

The PEG setup is divided into 2 parts.

1. A setup that needs to be done once on the machine.
2. Hook creation that needs to be done every time a new branch is created.

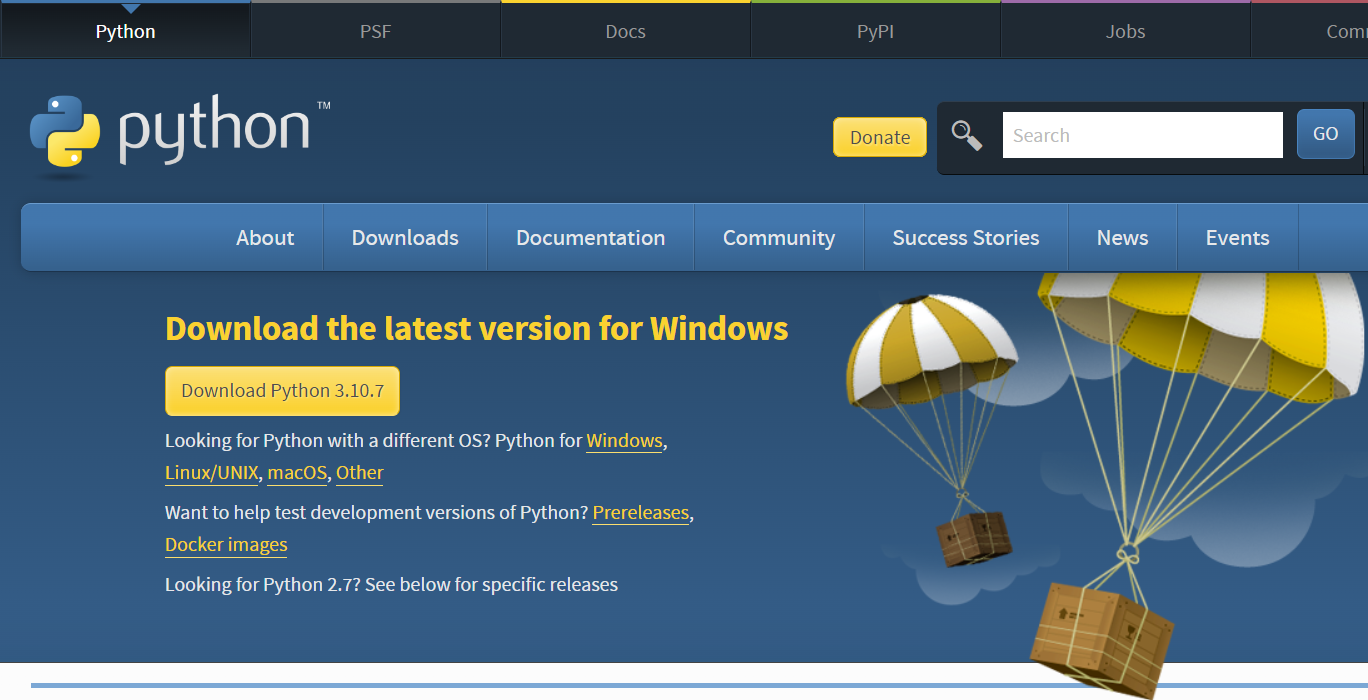
**Background:**

PEG project uses git’s own hooks to execute certain python scripts. Here, we’ll be using the pre-commit hook, which executes before a commit happens on the local branch. In the python scripts, we have coded the logic to check for SQL scripts that are not following our schematic style guide rules. To setup and execute the pre-commit hook with the python scripts, we will be using python package called pre-commit.

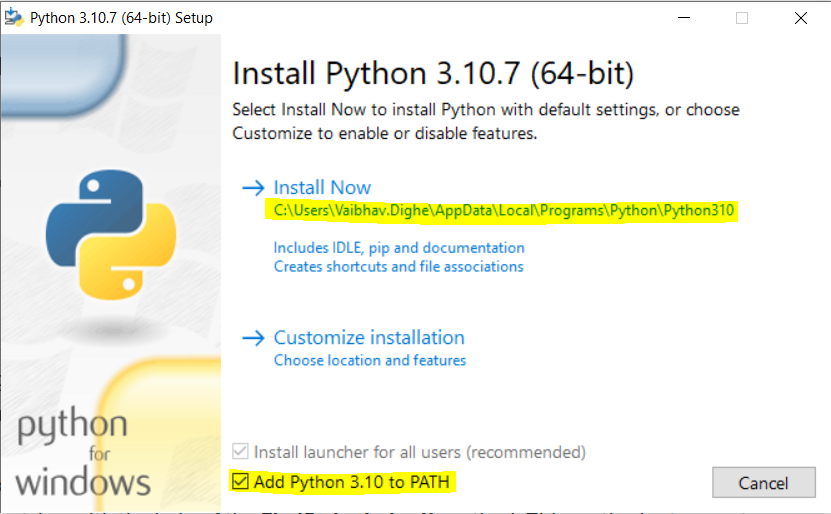
**One time setup:**

To use the pre-commit python package, we need to install python3, most of us are currently using Anaconda’s python environment which unfortunately is not supported with the pre-commit package.

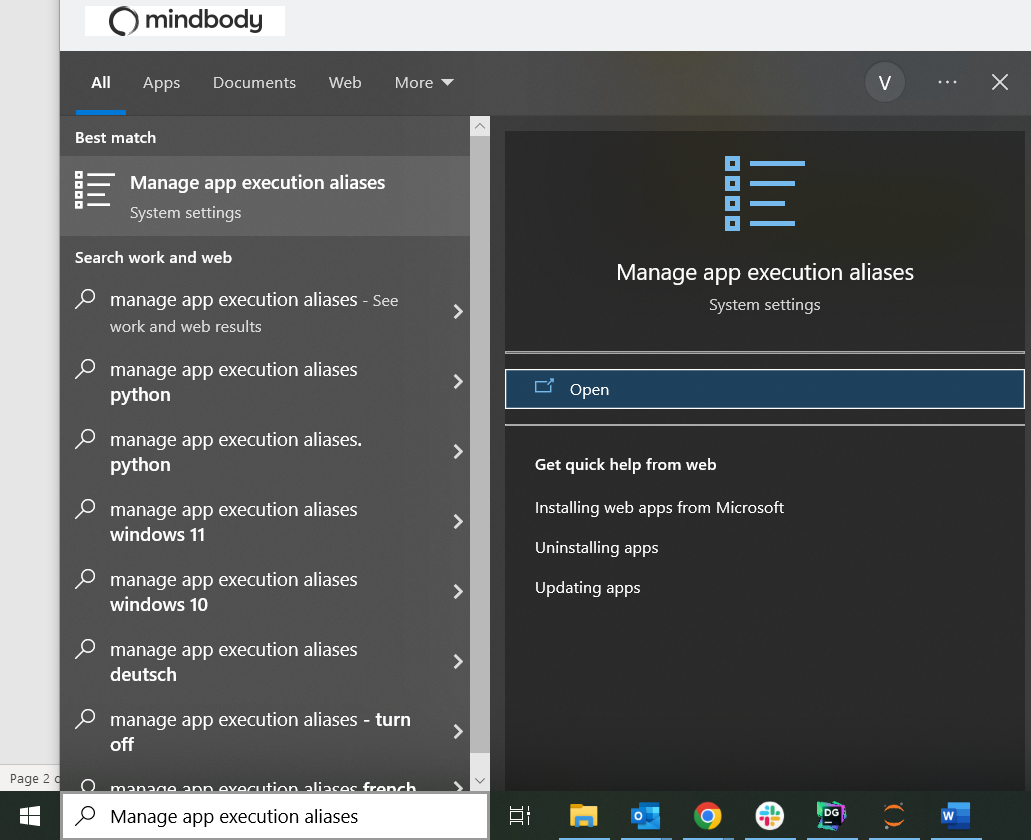
To install python3, visit <https://www.python.org/downloads/> and download the latest stable version of python3, which is found at the top on this page.



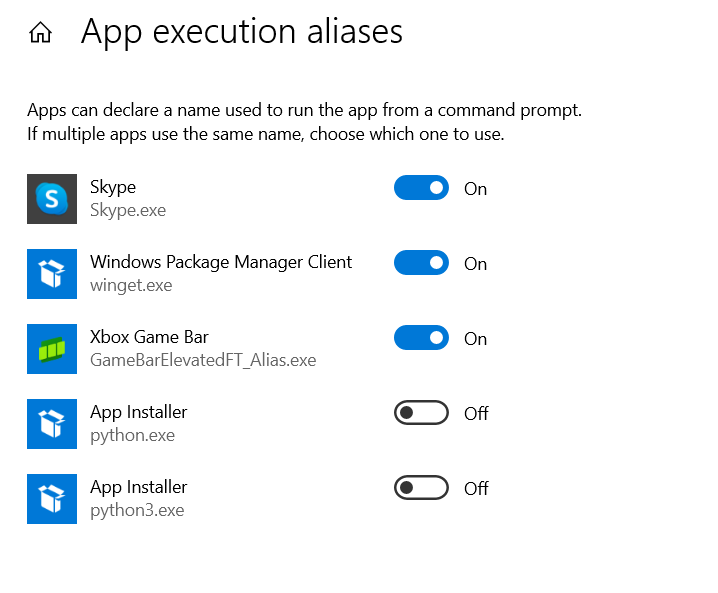
After downloading this setup, install it on your computer while making sure you add python to your PATH i.e., environment variables and note the file path where python is getting installed. Which will by default be your users folder *\Appdata\Local\Programs\Python\Python3xx* as shown in screenshot below. *(Noting down this installation path somewhere is a good idea as we’ll need this path a lot in further steps)*



After this installation is finished, search for *Manage App Execution Aliases* in your windows search box and open it.



In the app execution aliases setting, you’ll see two options of App Installer python.exe and App Installer python3.exe, turn off these 2 options. This will allow you to operate the Python3 environment that we just installed, from your windows command prompt.



Open your command prompt to check if the windows command line can access the python environment or not. Type in **python --version** to check the same.

Text

Description automatically generated

The command line should return the correct version of python that we just installed, in case this command doesn’t return python version or throws an errors saying python is not installed, you might need to restart your computer after python 3 installation and try the command again, it should work fine.

After you see the correct version of python on your machine through command line, it is time to install pre-commit python package.

To install the pre-commit python package type in the command prompt,

**pip install pre-commit**

This should install the latest stable version of this pre-commit package. You can check if the installation is successful or not by typing **pre-commit --version**

Text

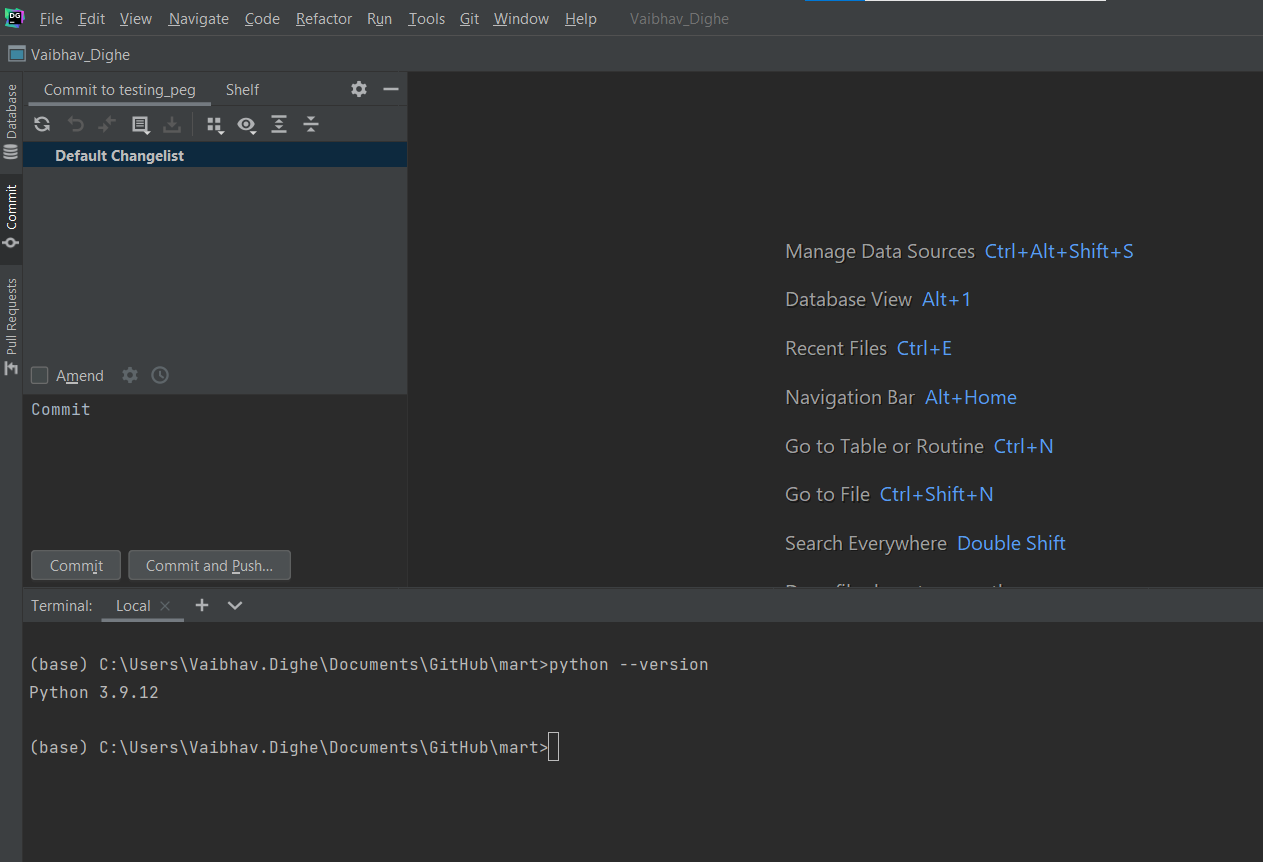
Description automatically generated

The installation of python 3 and pre-commit package in your python 3 environment is now complete.

Now, we’ll install the pre-commit python package in Anaconda’s python environment.

Open DataGrip and type in the DataGrip terminal **python --version**

This will return python version installed through anaconda, this version could be different that what we saw on windows command line, but that is fine.



We need to install the python pre-commit package on this environment as well, same commands from the DataGrip terminal will install it.

**pip install pre-commit**

To check if the pre-commit has been installed successfully from DataGrip, type in **pre-commit --version**

Graphical user interface, text

Description automatically generated

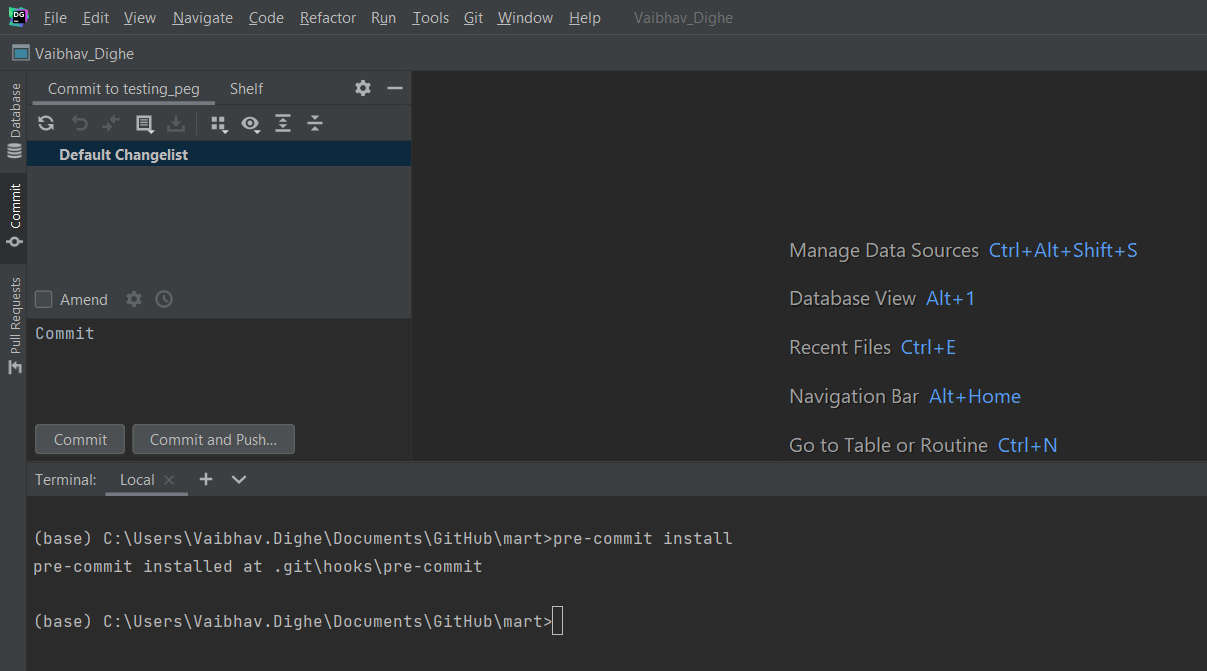
After this installation is done, we’re done with the 1 time setup that is needed for PEG execution.

**Hook creation for new branches:**

This part of the setup is something that we will need to do once per branch, i.e., after you create a new branch, before committing any files to it.

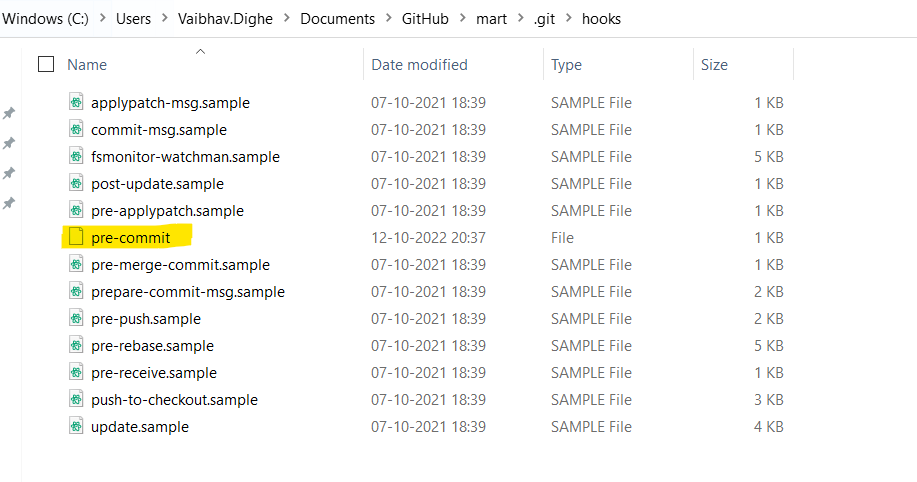
We will now be creating a pre-commit hook inside the .git/hooks folder on our local branch. Here, the pre-commit package will come in handy and create the hook for us from command line.

After you switch to the new branch, in DataGrip terminal type in **pre-commit install**



This command will create the necessary pre-commit hook file in your .git/hooks folder. The file is named *pre-commit* and it will not have any extension. To view this file you can go to your local mart folder and navigate to .git/hooks folder (make sure you have hidden files visible to view the .git folder). You can easily navigate to the local mart folder by going to the files panel in DataGrip and right click on the mart folder go to *open in -> explorer.*

Once you navigate to the .git/hooks folder, open the file named pre-commit (the file won’t have any extension).

****

Open this pre-commit file in your favorite text editor, you’ll see Anaconda’s python path in that file in front of INSTALL\_PYTHON variable. Change that path to the Python 3 path that we installed our python 3 in *(From the one-time setup shown above)*, as shown in screenshot below.

Example of the path: C:\Users\Vaibhav.Dighe\AppData\Local\Programs\Python\Python310\python.exe

Graphical user interface, text, application, email

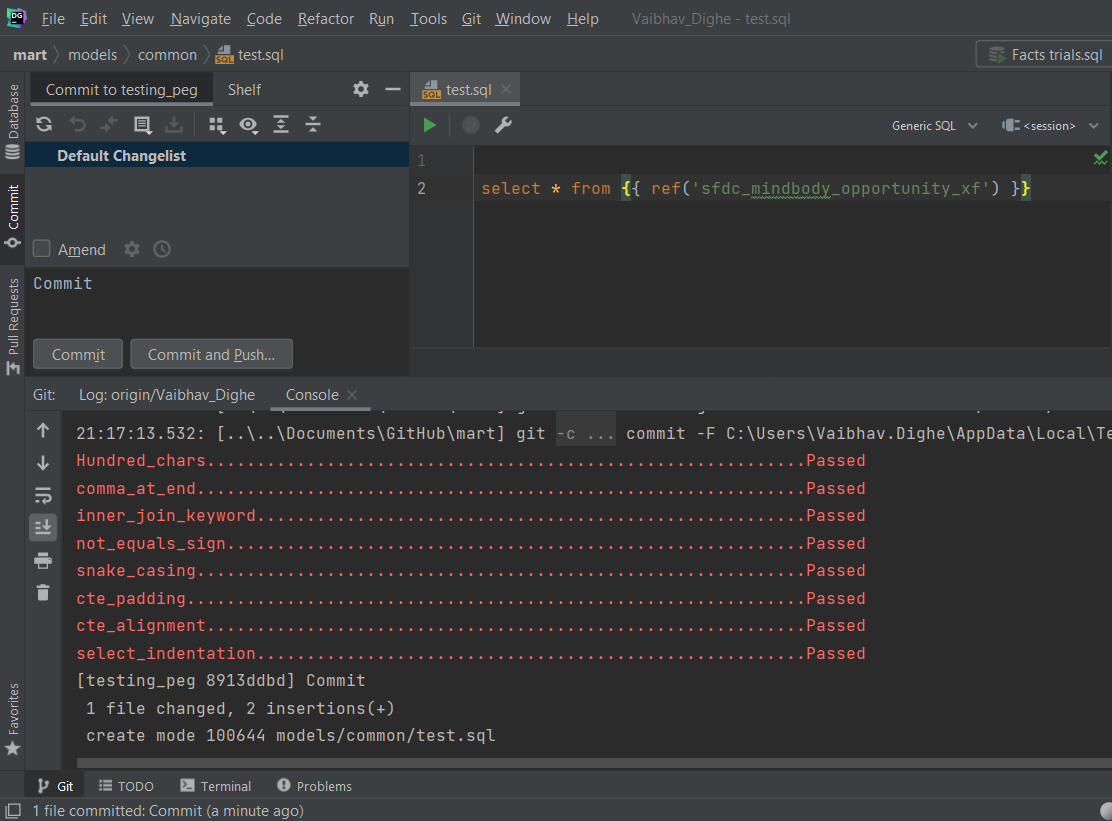
Description automatically generated

Once we change this path, we are good to go with the execution of the hook.

The pre-commit hook will use .pre-commit-config.yaml file from the mart folder and execute the python scripts mentioned in the .yaml file.

**Execution:**

Once the hook creation is done and the python path in the hook is changed, the hook will check all the files being committed to the branch for style guide errors.



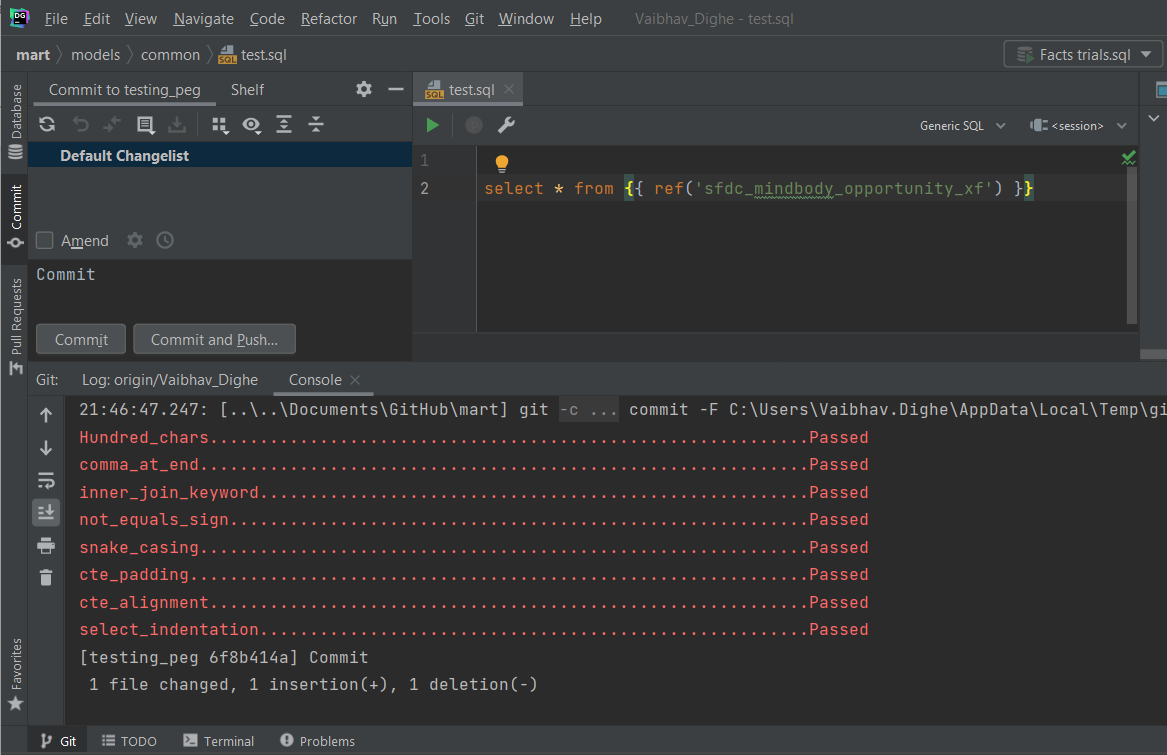
If the file that you’re trying to commit contains any style guide errors, the file won’t be committed successfully and the corresponding hook will show an error along with the line number in question.

As you can see in the example shown below, I tried to commit a file with a capital letter in it and the snake\_casing hook failed with appropriate error message.

Text

Description automatically generated

After I fix the file and commit it again, it will be committed without any error.



**Bypassing pre-commit check:**

If in case you are working with a file that might be an exception for the style guide rules and you want to commit the file anyway, you can do so in 2 different ways shown below.

1. To remove the pre-commit checks temporarily and committing the file without these checks.

To remove the pre-commit hook, we’ll need to put a command in the DataGrip terminal as,

**pre-commit uninstall**

This command will not uninstall the pre-commit package, but it will delete the pre-commit hook file which is responsible for the checks.

Graphical user interface, text, website

Description automatically generated

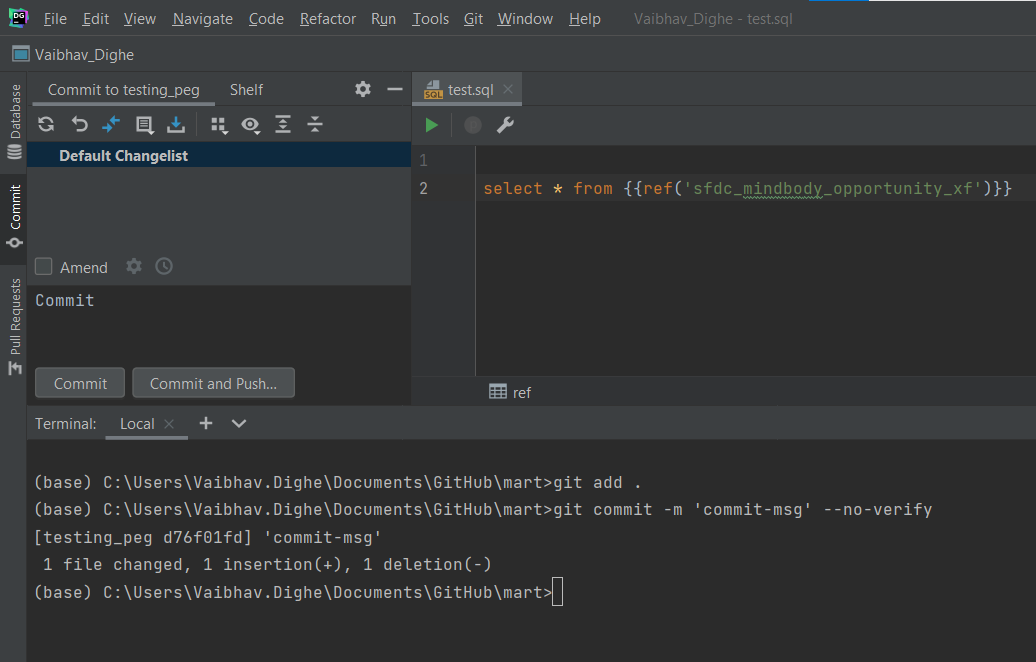
After the hook is deleted, any file can be committed to the branch without any restrictions or checks.

After committing a file with exception, if in case you would like to resume the pre-commit checks on further files, follow the hook creation for new branches guide from this document again to create the pre-commit hook again. *(Make sure to change the python path again after hook is re-created)*

1. Second way to commit the files without pre-commit checks is to commit them using DataGrip terminal and adding a **--no-verify** argument.

Use **git add .** command to add all new/changed files for staging, or use **git add filename** to add a specific file for staging.

Use **git commit -m ‘Commit message’ --no-verify** command to then commit the files added for staging without making them go through the pre-commit checks. Here, the **--no-verify** argument will make these files bypass the hook.



**Few things to note:**

1. The python scripts responsible for these style guide checks are kept in the scripts folder of our mart directory.
2. If commit is throwing any error that you’re not able to understand and the files need to be committed urgently, 1st method mentioned in the *Bypassing pre-commit checks* section of this document should do the trick.
3. The setup that is done after creating a new branch may sometimes stay the same even after you create a new branch/change the current branch which might allow you to skip the hook creation for new branch, but this has been observed as being a little inconsistent which is why it is a good practice to create the hook again for a new branch.