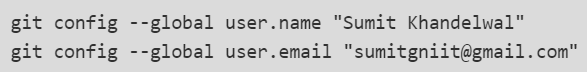
**Working with GIT Commands**

Login on <https://gitlab.com> using Gmail credentials.

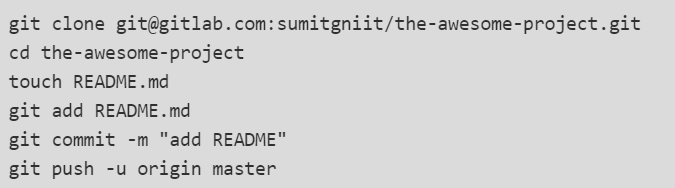
Create new project – **the-awesome-project**

Set the visibility level to private

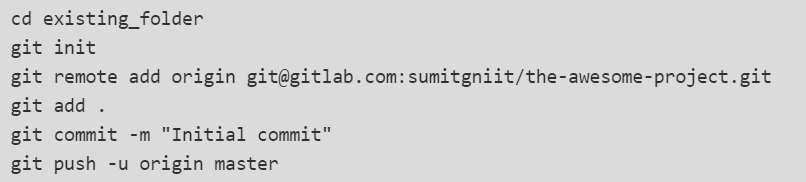
Configure Git global setup –



Create the new repo –



Try to push an existing folder –

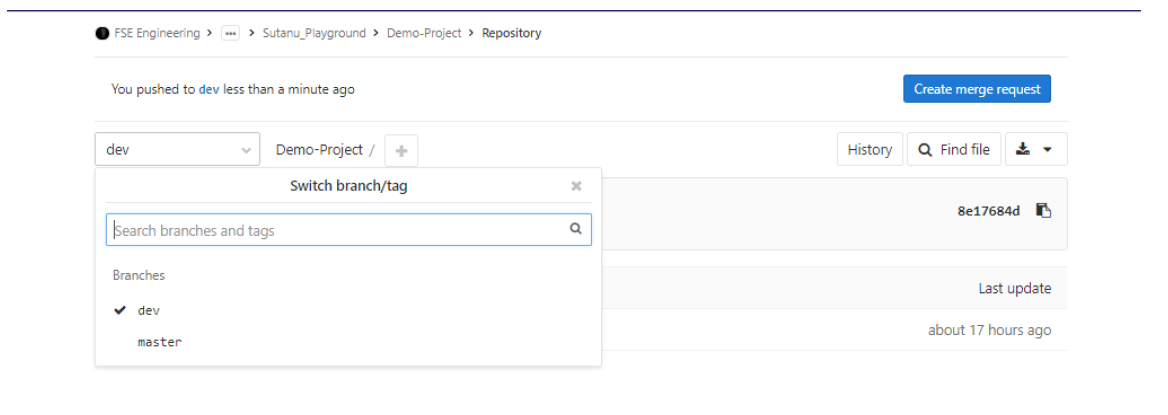
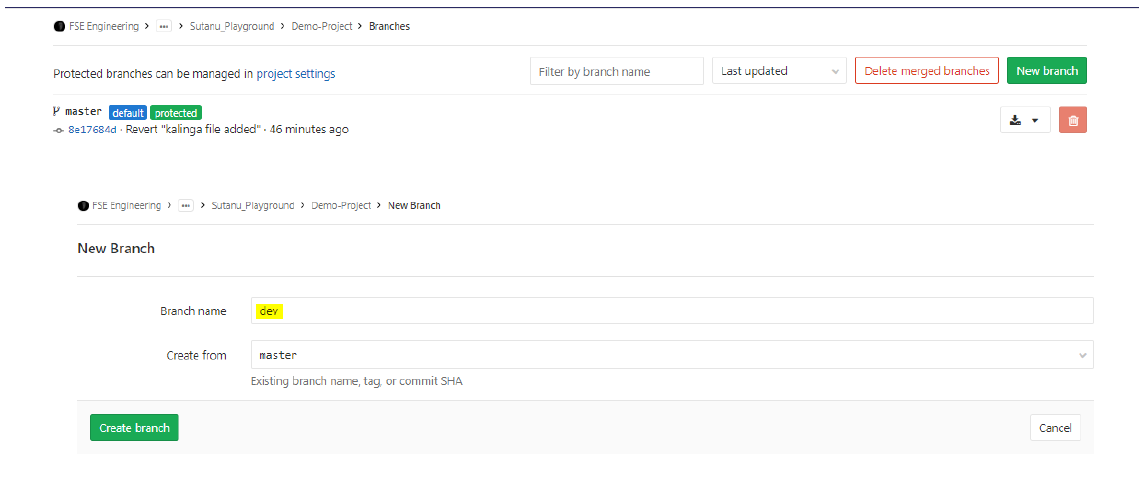


Now your code is pushed to gitlab.

Now go to your gitlab project and refresh to see the readme file.

**Git Commands**

1. **Clone**: To clone an existing project copy the ssh code given on top of your project. Clone the project on gitlab in your local system.
2. **Add**: After cloning the project if you make some changes and now you want to make the same changes reflect on git project
   1. “*git add filename*” to add a specific file.
   2. “*git add .”* to add all the files.
3. **Commit**: This command stores the current content on index in a new commit along with a commit message from the user.
   1. With the commit user can specify a message using the option **–m** or **–am**.
   2. If the user selects –m option then only the files he made changes will reflect the message in git repository.
   3. And if the user chooses to use –am option then all the files will show the given message in git repository.
4. **Push** : After the commit the changes has to pushed to git for that we use “git push” command.
   1. EX: for example lets add a file names “kalinga.txt” to our local project now use the 3 commands to push the changes to git repository.
   2. Now if we go the git and refresh we will find the change with the commit message.
5. **Revert** : If we push some unwanted change into git there is always a way to go back to any previous version using revert. We can revert any change directly from git repository.
   1. Go to commits -> click on the commit you want to revert -> choose revert option from options drop down -> uncheck the new marge request check box. -> click on revert.
   2. Git treats revert as a commit too.
6. **Move Rename Delete :** Locally if we move delete or rename a file and push to git then all the changes will reflect to git.
7. **Create Branches** : Branches makes work in a very big group very easy. In a big project there are a lot of different works which are completely independent or separate from each other so different work can be done in different branches simultaneously without effecting others work.
   1. Lets take an example: Suppose in a project there is development, research, bug fixing and testing going parallel which are not dependent on each other. So, we can have different branches for all these works and the assigned team can work without being dependent on other work.
   2. To create a new branch, follow the steps:
      1. Go to branches -> click on new branch -> give a name to the branch -> click on Create branch -> your branch is created with given name.
      2. Newly created branch will hold the content of the branch which you select in the field create from.



Now in the branches dropdown the dev branch is visible.

1. **Merge, Conflict and Resolve :** Merge conflict occurs when someone tries to push into a change into git without having the latest content.

Let’s say two people made changes into a same file and the second one tries to push his/her code. As he/she does not have the updated content it will generate a marge conflict. And git bash will prompt to pull the latest code. At the time of pulling the code it will generate a marge conflict locally.

Let’s create a text file named “kalinga” and generate a marge conflict.

To resolve marge conflict, we need to choose which changes to keep or both to keep and remove the head and other symbols and then push it back to the git repository. Then, we can push the file with the current changes.

1. **Forking a project:** Fork creates a copy of the project. By forking we can work on a project without disturbing the original project.
   1. To fork a project, go to the *project* and click on **fork**.