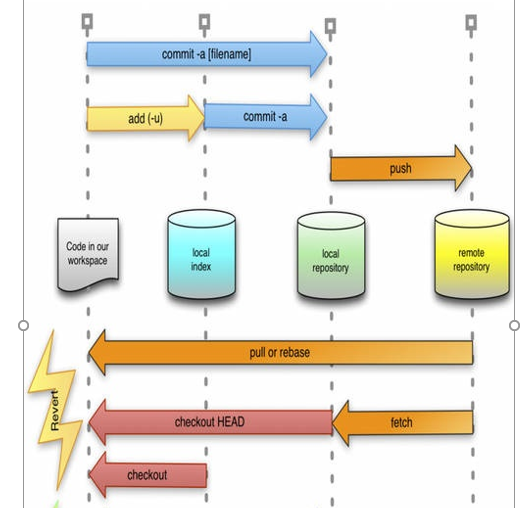
**GitHub Implementation for the Chemours Microsoft Applications**

**GitHub:**

GitHub is a web-based hosting service for software development projects that use the Git revision control system.

**Git Revision Control System:**



**Distributed Version Control System:**



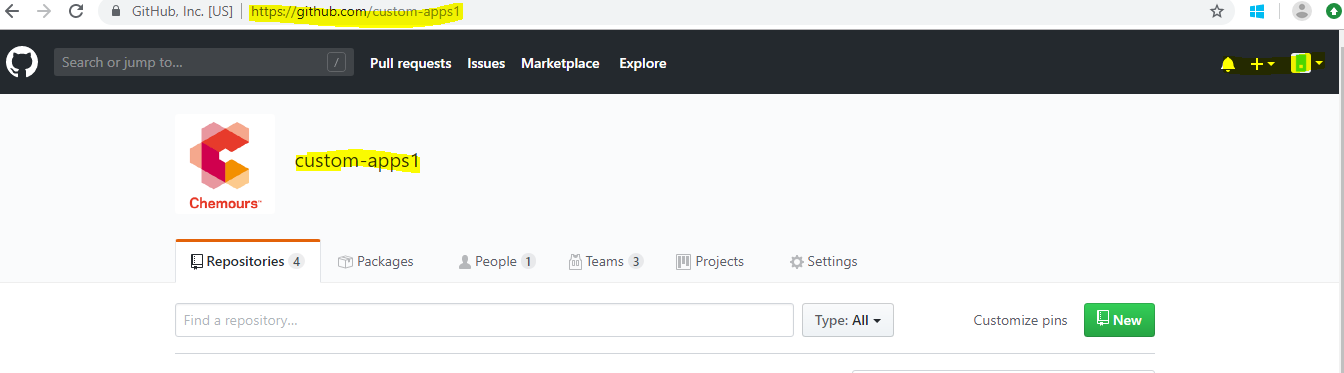
**Setup:**

1. GitHub setup
2. Local Environment setup

**A. GitHub setup:**

1. Create GitHub Admin/Super User Account.

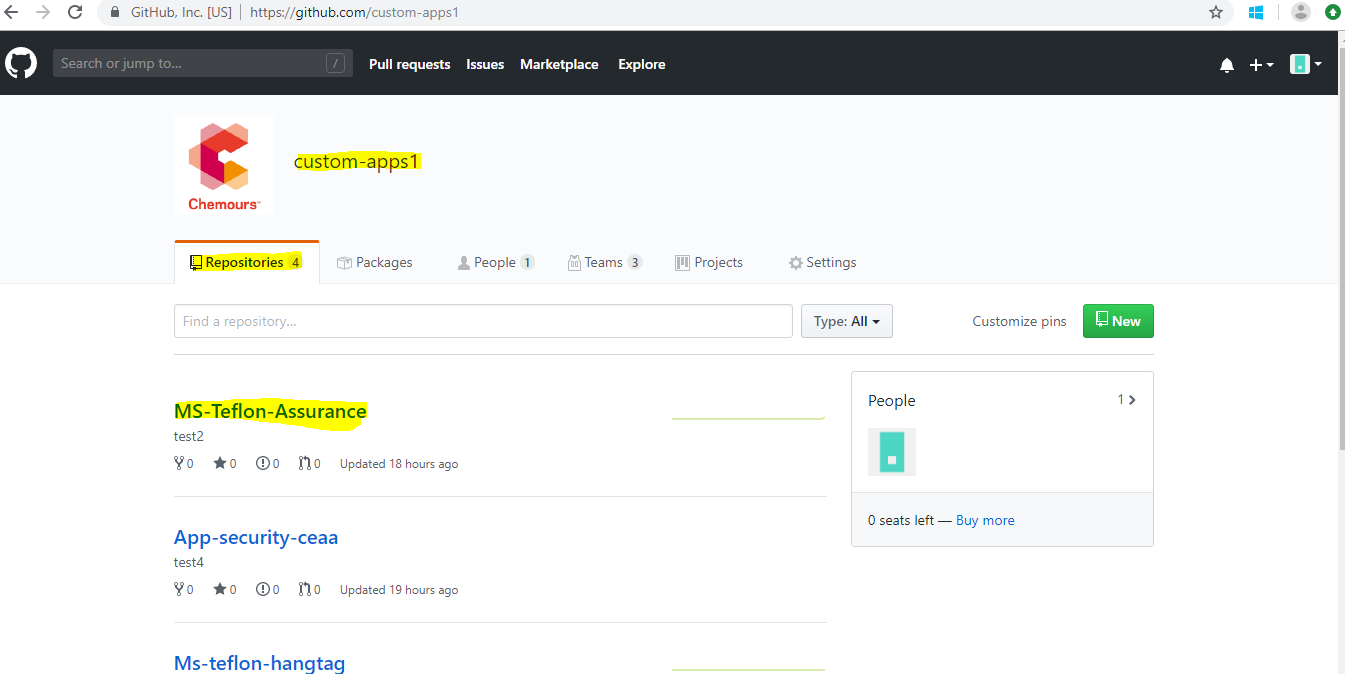
2. Create Organization for Microsoft Applications “Custom-Apps”



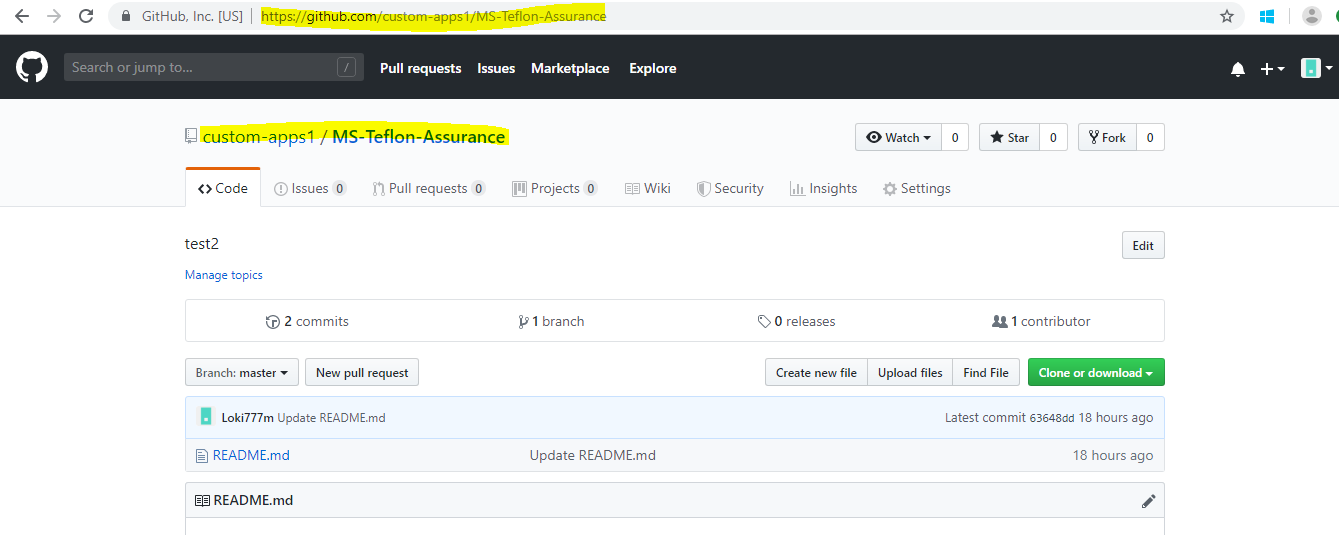
3. Create Repositories.

Create one repository per application.

|  |  |
| --- | --- |
| **Microsoft Teams (Application)** | **Repository Name** |
| custom-app-Legal-business-ethics | Legal-Business-Ethics |
| custom-app-MS-Teflon-assurance | MS-Teflon-Assurance |
| custom-app-MS-Teflon-hangtag | MS-Teflon-Hangtag |
| custom-app-security-ceaa | Security-CEAA |
| custom-app-me-ecapex | Me-Ecapex |
| custom-app -fi-smart | Fi-smart |
| custom-app -fi-feed | Fi-feed |
| custom-app-sourcing-eppp | Sourcing-EPPP |
| custom-app-ms-krytox | MS-Krytox |

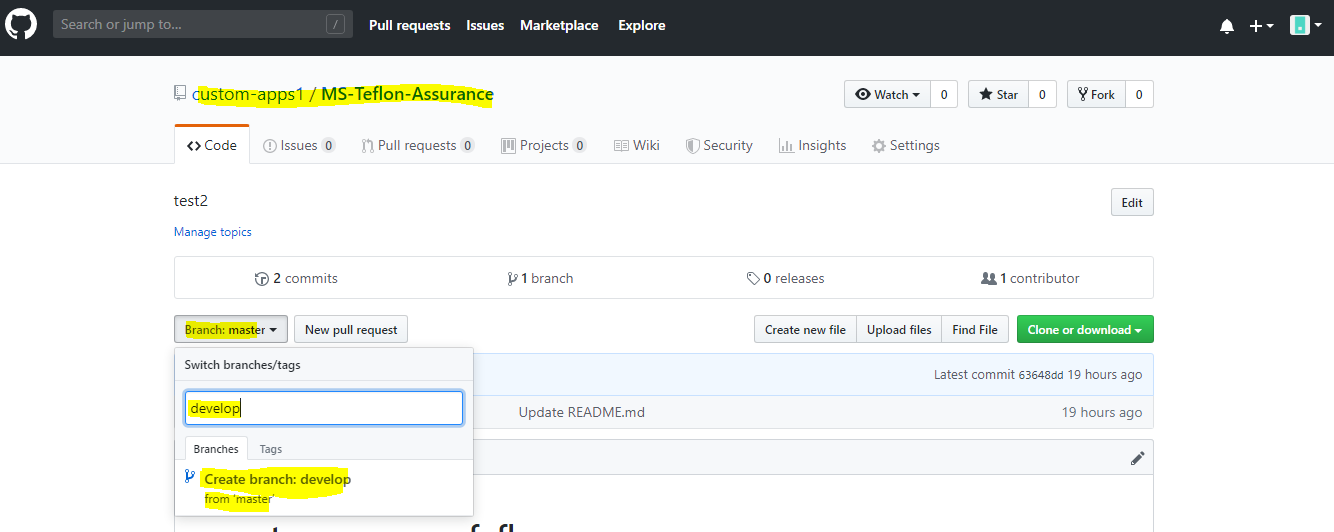


Repository looks like this and can upload all code files (Production) related to the application. Default branch will be the master branch. **Origin/master** branch is the main branch where the source code of HEAD always reflects a production-ready state.



4. Create development branch

Create development branch “**develop**” in parallel with master branch.

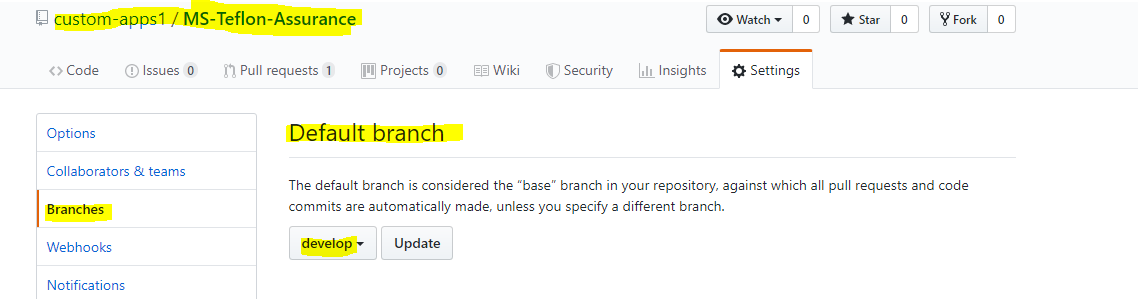


* The Origin/develop will be the main branch where the source code of HEAD always reflects a state with the latest delivered development changes for the next release.
* When the source code in the develop branch reaches a stable point and is ready to be released, all of the changes should be merged back into master somehow and then tagged with a release number.
* Therefore, each time when changes are merged back into master, this is a new production release by definition.

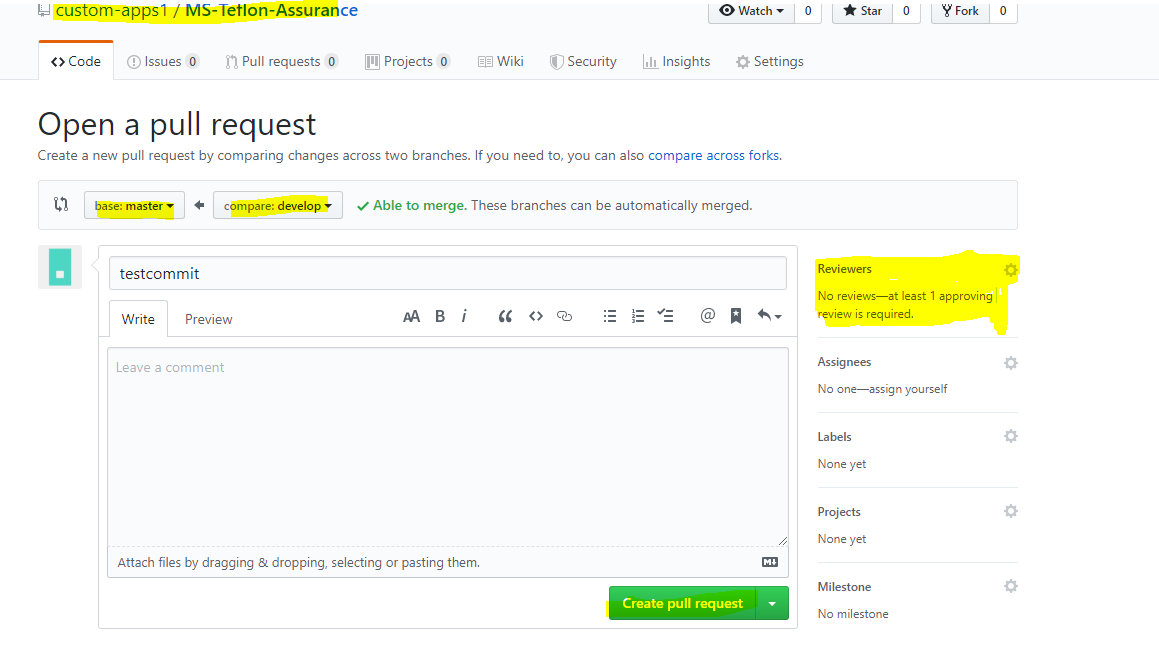
5. Create QA branch with name “QA” from the develop branch and upload the source code changes related to QA source code.

* QA branch will be the main branch and having QA ready source code.
* Each time when we have to make changes in QA code, we have to create new branch from the QA as “QA\_V1.0.0” (as per the versioning) and make the changes, commit and merge the branch with QA branch.

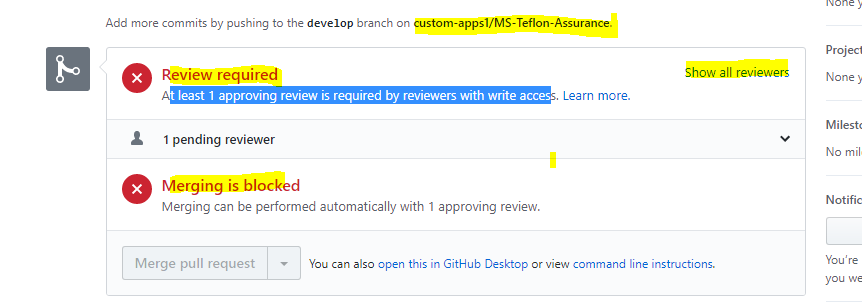
6. Setup branch protection rules for “**master**” and “**develop**” branches for safe merging, reviewing and approving before merging.



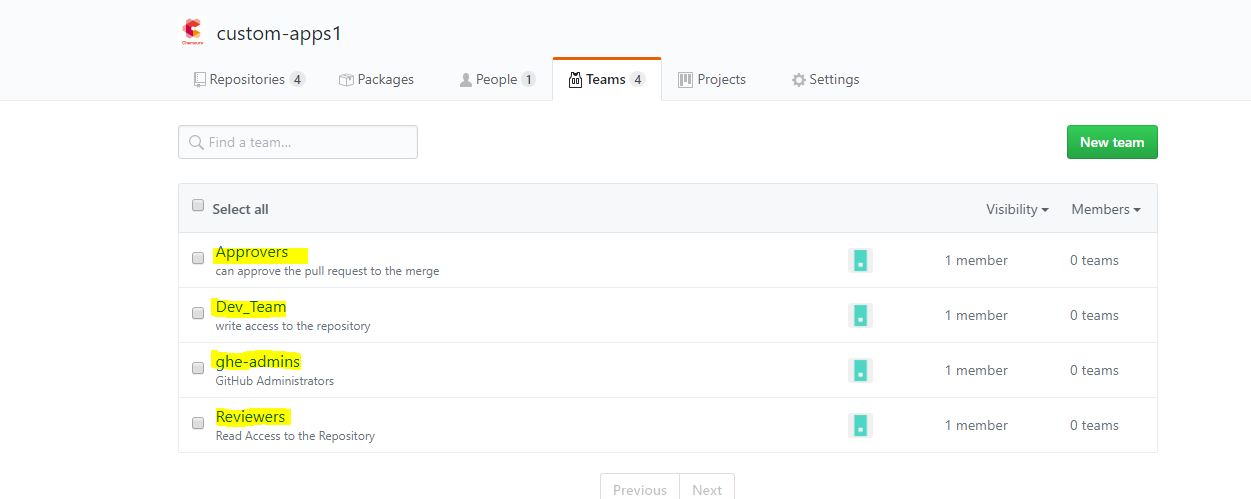
Create branch protection rules.



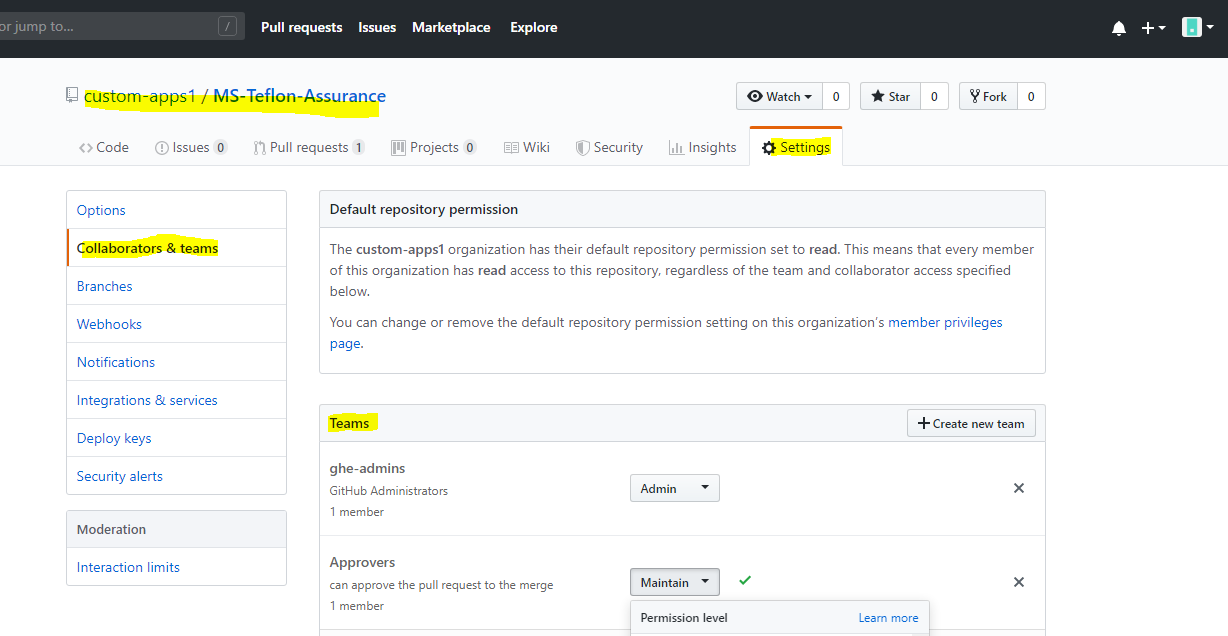
Eg:

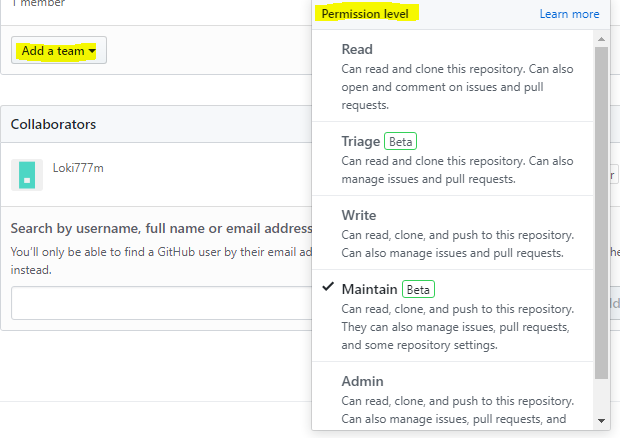


7. Create Team for the organization and adding members to the team. Access controls.



8. Add teams to the repositories and assign the access.





8. Setup email addresses to receive notifications when push events are triggered.

**B.** **Local Environment setup**

1. Install GIT and GIT Bash on the user’s local desktop.

2. User Configuration in Git

**a**. **Setup email address, user name etc. globally**.

git config --global user.name "Lokesh Makkineni"

git config --global user.email "Lokesh.Makkineni@chemours.com"

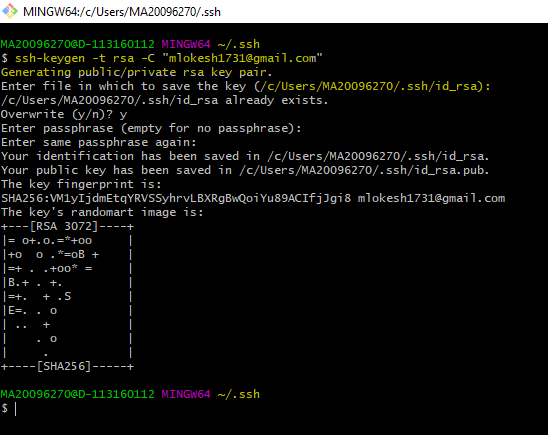
git config --global merge.conflictstyle diff3

**b**. **SSH Key**

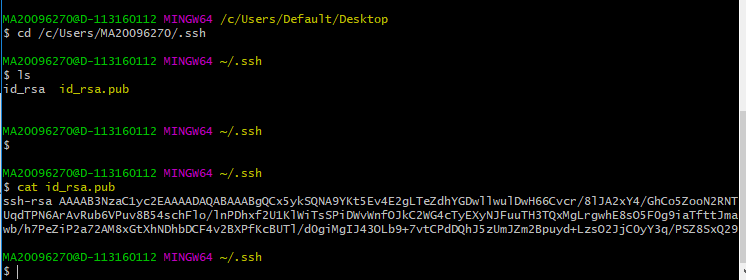
SSH Keys are the way to identify trusted computers without involving passwords. We can generate the SSH key and add the public key to GitHub account.

Generate the SSH Key.

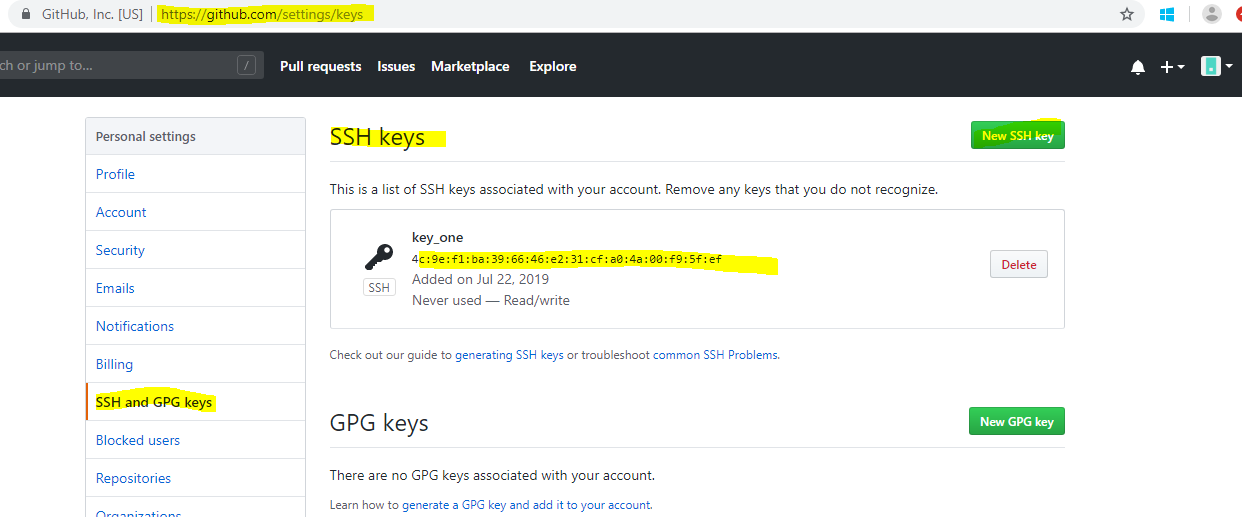
ssh-keygen –t rsa –C “Lokesh.Makkineni@chemours.com”



Copy the public key



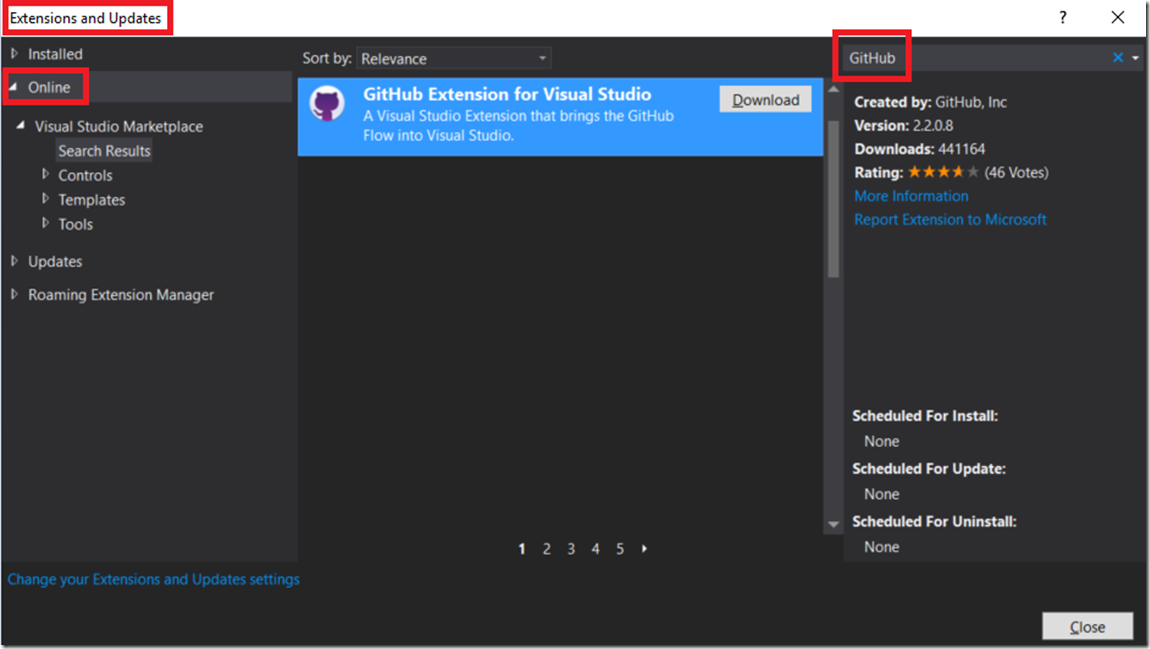
Add the public key on GitHub Account



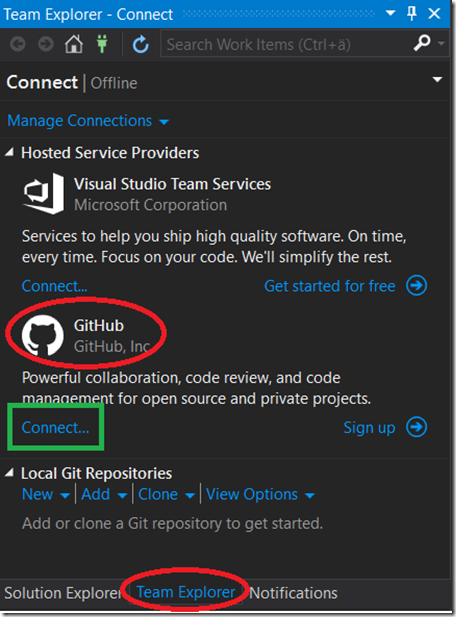
3. IDE Integration –Visual Studio

a. Install GitHub for Visual Studio

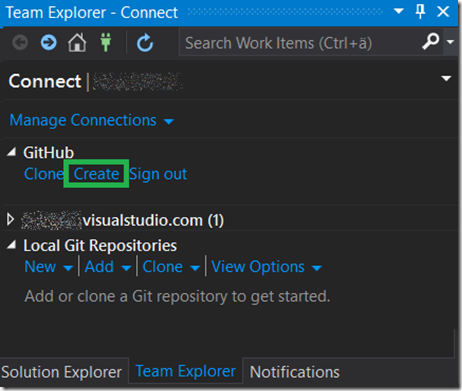
In Visual Studio select Tools -> Extensions and Updates, which open the window. Search for GitHub extensions, Download and install the extension, a restart the Visual Studio.



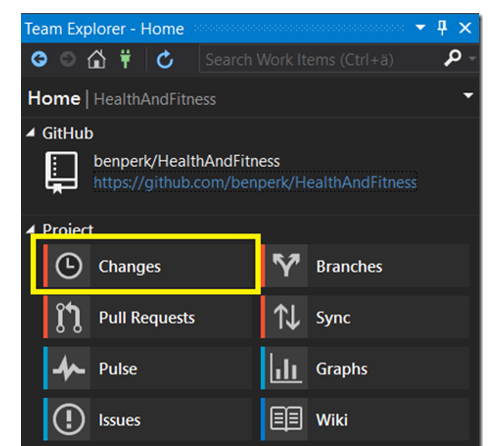
b. Connect to the GitHub repo and login



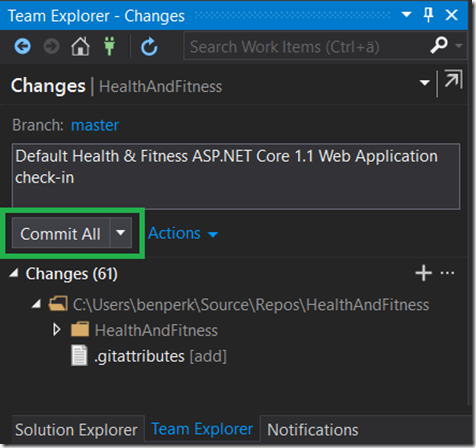
c. Clone or create repository.



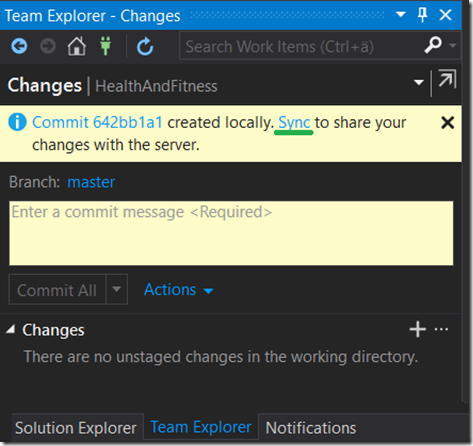
d. Add the source code to GitHub

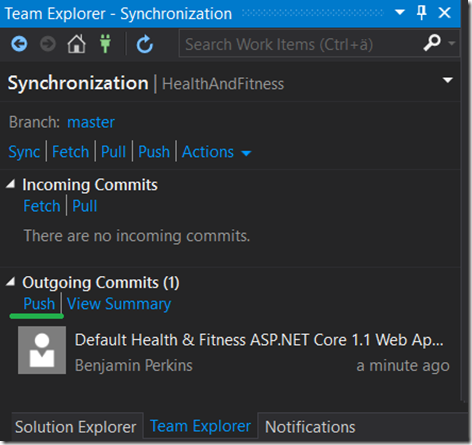


e. Commit the changes and changes will be added to the local repository.



f. Push the changes to GitHub repository





**Key Features of GitHub Administration:**

* Properly configure GitHub Enterprise Cloud account to meet our organization’s needs.
* Improve the developer experience by selecting appropriate organization settings.
* Identify the recommended options and configurations to reach your desired outcomes.
* Enable protected branches, issues, GitHub Pages etc.
* Demonstrate daily and advanced user behavior on GitHub.
* Locate metrics from your GitHub Enterprise organization.
* Audit critical activities performed on your GitHub Enterprise Cloud account.

**Responsibilities of GitHub Administration:**

* Working on GitHub
  + Repository permissions and settings
  + GitHub Pages
  + Create branches and Pull Requests
  + Configure protected branches
  + Configure required status checks
* Security in the cloud
  + Security alerts
  + User privacy
  + Licenses
  + GitHub Marketplace
  + Integrations
* GitHub Enterprise site administration.
* Audit account activities.
* Select repository and organization settings.
* Administer users.
* Manage dormant users.
* Organizations and teams.
* Use web hooks.
* Advanced user behavior and administration.

**How Importance GitHub in DevOps?**

