**GIT REPOSITORY CREATION(Set Up)**

Step 1:- Create a GIT account at <https://github.com/>

Step 2:- Go to right corner drop down (account) and select >Your Repository> Click New

Provide a Repository name then click on >Create Repository.

Step 3:- Download and Install GIT on your machine from <https://git-scm.com/download/win>

<http://gitimmersion.com/lab_01.html>

Step 4:- To setup GIT in your System type following commands in command prompt.

git config --global user.name "Your Name"

git config --global user.email "your\_email@whatever.com"

Step 5:- Copy your GIT Repository URL from Clone or Download and clone into your system folder using:-

git clone https://github.com/bobleeswagger08/Hello-World.git

Change to the clone directory

>cd Hello-World

<https://stackoverflow.com/questions/8775850/how-do-i-add-files-and-folders-into-github-repos>

Step 6:- To Load your project or files into just created repository:-

>git add <filename> or

>git add <foldername> or

>git add \*

Note:- adding files is synonyms to staged changes you can type >git reset to remove files from staged changes.

Step 7:- To commit changes type:-

>git commit –m “commit message”

Step 8:- Finally use:-

>git push –u origin master

(to push the remote branch master for the repository origin.)

Running a git pull origin master to ensure you have absorbed any upstream changes

If you do not want to merge the upstream changes with your local repository, run git fetch to fetch the changes and then git merge to merge the changes. git pull is just a combination of fetch and merge.

**Some Command Utility:-**

**>**git remote

It lists the shortnames of each remote handle you’ve specified. If you’ve cloned your repository, you should at least see origin — that is the default name Git gives to the server you cloned from.

>git remote -v

Which shows you the URLs that Git has stored for the shortname to be used when reading and writing to that remote

**>**git status

When you make modifications run git status which gives you the list of files modified, add then using git add \*

**Git Branching**

[**https://guides.github.com/activities/hello-world/**](https://guides.github.com/activities/hello-world/)

[**https://www.atlassian.com/git/tutorials/using-branches**](https://www.atlassian.com/git/tutorials/using-branches)

The git branch command lets you create, list, rename, and delete branches. It doesn’t let you switch between branches or put a forked history back together again. For this reason, git branch is tightly integrated with the [git checkout](https://www.atlassian.com/git/tutorials/using-branches/git-checkout) and [git merge](https://www.atlassian.com/git/tutorials/using-branches/git-merge) commands.

* git branch

:- lists all the branches in your repository. This is synonymous with git branch –list.

* git branch <branchname>

:- Create a new branch. This does not check out the new branch.

* git branch –d <branchname>

:- Delete the specified branch. This is safe operation in that Git prevents you from deleting the branch if it has unmerged changes.

* git branch –D <branchname>

:- Force delete the specified branch, even if it has unmerged changes. This is the command to use if you want to permanently throw away all of the commits associated with a particular line of development.

* git branch –m <branchname>

:- Rename the current branch(master) to <branchname>.

* git branch –a

:- List all remote branches.

git branch crazy-experiment

The repository history remains unchanged. All you get is a new pointer to the current commit:

Note that this only *creates* the new branch. To start adding commits to it, you need to select it with git checkout, and then use the standard git add and git commit commands.

Successful Branching Model

<https://nvie.com/posts/a-successful-git-branching-model/>

<https://stackoverflow.com/questions/14865283/proper-git-workflow-scheme-with-multiple-developers-working-on-same-task>

* Have 1 master and 1 develop branch
* Have feature branches spawn off of develop branch
* Every time you have version ready for **QA** to test, merge into develop
* Have release branches spawn off of develop branch
* Make bugfixes into release branches
* When you have version ready for **QA** to test, merge into develop
* When you have version ready for **PRODUCTION**, merge into master, and create a tag for it

For Visual Studio Code

<https://code.visualstudio.com/docs/editor/versioncontrol>

We can type (Ctrl+Shift+P) to open command palette and type GIT all the methods will be available for that Git repository operation.

To view the history of the project

>git log

To remove a folder from GIT repository

> rd /s /q .git

All the data Git uses for information is stored in .git/, so removing it should work just fine. Of course, make sure that your working copy is in the exact state that you want it, because everything else will be lost. .git folder is hidden so make sure you turn on the Show hidden files, folders and disks option.

From there, you can run git init to create a fresh repository.

git submodule add https://github.com/NishanAsim/RestSamples.git customer-app

git remote add origin https://github.com/{username}/{reponame.git}  
git remote -v  
then run git pull origin master and thats it

* In the command-line, navigate to your local repository.
* Ensure you are in the default branch:  
  git checkout master
* The rm -r command will recursively remove your folder:  
  git rm -r folder-name
* Commit the change:  
  git commit -m "Remove duplicated directory"
* Push the change to your remote repository:  
  git push origin master

Create a subfolder:-

Git add foldername/\*

<https://stackoverflow.com/questions/12258399/how-do-i-create-a-folder-in-a-github-repository>