**BRANCHING:**

Create a branch on master replica, as the changes should not be done on master

Master should always has to be deployable

Branch is exact copy of master

**MERGE:**

Commits the branch

**MERGE CONFLICTS:**

When there is a change in same line in 2 branches

Try git status

And open the file in editor

Keep the change you want to keep

(Please find the below Git Branch diagram attached)

****

**GIT BRANCH**

Before creating a new branch, we need to keep our master branch clean, by clean I mean without any changes, like that you can create at any time a branch from your master. Each time that you want to commit a bug or a feature, you need to create a branch for it, which will be a copy of your master branch.

When you do a pull request on a branch, you can continue to work on another branch and make another pull request on this other branch.

Before creating a new branch, pull the changes from upstream. Your master needs to be up to date.

**Create the branch on your local machine and switch in this branch:**

$ git branch release

**Change working branch:**

$ git checkout release

**Push the branch on github:**

$ git push origin release

*When you want to commit something in your branch, be sure to be in your branch. Add -u parameter to set upstream.*

**You can see all branches created by using:**

$ git branch –v (local branches)

$ git branch –r(remote braches)

**Add a new remote for your branch:**

$ git remote add origin <http://github.com/Inmar/repo.git>

**Verify the new remote**

$ git remote –v

**Note:** *If the remote name is already exists then you can rename the current remote name*

**Push changes from your commit into your branch:**

$ git push origin <http://github.com/Inmar/repo.git>

**Update your branch when the original branch from official repository has been updated:**

$ git fetch origin

**Then you need to apply to merge changes, if your branch is derivate from develop you need to do:**

$ git merge origin/develop

**Delete a branch on your local filesystem:**

$ git branch –d release

**To force the deletion of local branch on your filesystem:**

$ git branch –D release

**Delete the branch on github:**

$ git branch push :release

**TAGGING:**

Tags are a simple aspect of Git, they allow you to identify specific release versions of your code. You can think of a tag as a **branch that doesn't change**. Once it is created, it loses the ability to change the history of commits.

Listing the tag:

$ git tag -a 'Release\_1\_0' -m 'Tagged basic string operation code'

Search for tags that match a particular pattern:

$ git tag –l

Push your tags to your remote repo:

$ git push origin –tags (multiple tags) / single tag then $ git push origin 'Release\_1\_0'

**MERGE**

Merge master into the development first so that if there are any conflicts, I can resolve in the development branch itself and my master remains clean.

(on branch development)$ git merge master

(resolve any merge conflicts if there are any)

$ git checkout master

$ git merge development (there won't be any conflicts now)

If we want to keep track of who did the merge and when, we can use --no-ff flag while merging to do so. This is generally useful only when merging development into the master (last step), because we might need to merge master into development (first step) multiple times in our workflow, and creating a commit node for these might not be very useful.

$ git merge --no-ff development

**Merge Conflicts:**

When we do git push/pull or git merge, we end up with conflicts.

When there is a File in the Remote Repository which is been accessed and worked by 2 associates named as A and B. When a person A and B makes a changes for the same file and submits push request to the Remote File then occurs conflicts ie., when A associates made the changes and submits the push request (it will successful) and later when B associates submits push request of the same file with his/her changes, then we encounter merge conflicts with the below error:

!rejected master 🡪master(fetch first)

**Error:** failed to push some refs to ‘git@github.com.satishgudey/file.git. Updates were rejected because the remote contains work that you do not have locally. Thiss is usually caused by another repository pushing to the same ref. you may want to first integrate the remote changes (eg., ‘git pull …’) before pushing again.

**Resolving the Merge Conflicts:**

Now instead of disturbing master branch as it has production deployable code, I will use ‘Rebase’. What Rebase will do is that it will always have an initial commits that will come from central repository, it will add up associate B commits on top of that workflow (or) working tree that shows all commits.

Now associate B will perform the below commands to overcome this conflicts:

1. Need to check the status to find out where the conflicts has occurred

**$ git pull –rebase origin master (or)**

**$ git status**

1. Now need to see the changes made by both the A and B associates by opening the merge tool

**$ git mergetool**

Note: Mergetool will show original file (master), and changes made by both the A and B associates on the top and at the bottom it will show the differences. Now associate B needs to decide whether to keep the changes or delete and save it.

1. Now associate B will perform Rebase with continue to apply the changes to the central repository.

**$ git rebase --continue**

1. Now associate B can push the changes to central repository

**$ git push origin master**