**Branching Strategies**

Let us suppose I have two branches which are in sync….

1. Master
2. Code(develop)

If we want to develop a new feature we have to create a branch from code branch.

$git checkout -b Features/feature1 code

$Vim feature.txt

$git add .

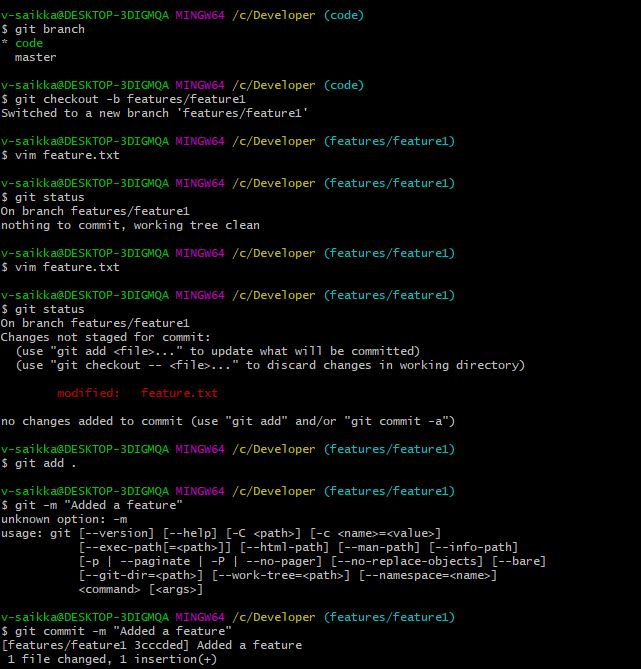
$git commit -m “message goes here”

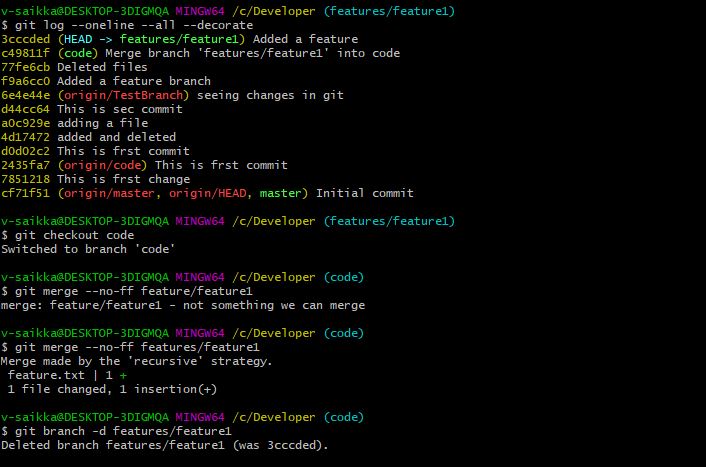
Now navigate to code branch……………..

$git merge –no-ff features/feature1

Now we merged all the changes made in feature branch in to develop code branch now we can delete the feature branch.

$git branch -d features/feature1





**Release Branch:** Support preparation for new production release(last minute fixing bugs & small changes before release)

The release branch should be created from code branch and merge back in to code branch then to master.

Naming should be (release-v1.0)

After merging from Release🡪code🡪Master it becomes production ready

$git checkout -b release-v1.0 code

$vim release.txt

$git add .

$git commit -m “Changed Version”

Now ready to go to production but before that we have to do two things

1. Go to master and merge all changes from release

$git merge –no-ff release-v1.0

Once the merge files have been merged it is necessary to tag the merge for a release.

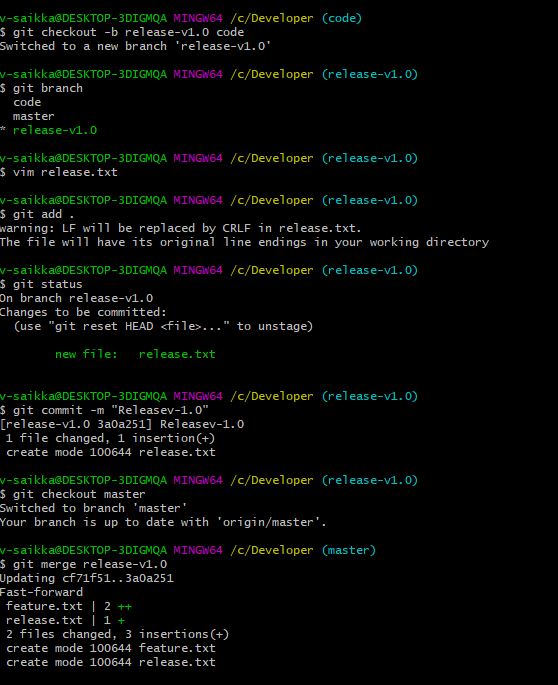
$git tag -a version-1.0

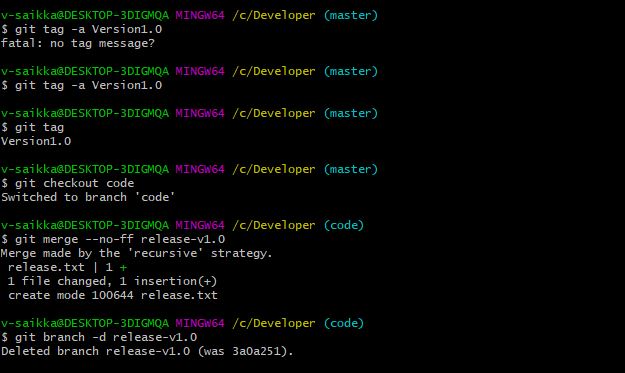
$git tag

Now merge the changes back to the code by switching to code branch.

$git merge –no-ff release-v1.0

$git branch -d release-v1.0





**Hotfix Branch:**

Very much like release branch unplanned fix bug.must merge back in to develop and master.

$git checkout -b hotfix-v1.0.1

$vim hotfix.txt

$git add .

$git commit -m “Message”

Now after fixing the bug and commiting it follow two steps

1.merge hotfix to master

2.merge hotfix to code

$git checkout master

$git merge –no-ff hotfix-v1.0.1

Master has all the changes now.

$git tag -a V1.0.1

$git tag

$Git checkout code

$Git merge –no-ff hotfix-v1.0.1

$git branch -d hotfix-v1.0.1

