**Working with Version Controller**

Till now we worked on Git with master branch, just we pull the code from master branch and developed code in master branch and pushed, with this we faced lot of problems like DB update and we can’t fix bug to the customers in delivered code.

To overcome to this problem, we had to work with branches, for this we have to do small practise with one dummy project.

I created one sample repository in Git, using this repository we practice Git version control

with branches and versions(tags).

First create one new folder in your pc.

Go to that path using git command prompt

* git clone https://github.com/nageshwar-mallepally/git-excercise.git

now the code is cloned into your pc.

Open the cloned project in intellij

In intellij bottom right corner we can see the branches Git:master. Click on master

Here we can see the branches of local and remote, if we need we can create new local branches here.

**Git Setup**

Create a **new Git repository** from an existing directory:

* git init

**Clone** **a repository** (local or remote via HTTP/SSH):

* git clone https://github.com/nageshwar-mallepally/git-excercise.git

To enter in branch

* git checkout –track -b <branchname>

After you checkout in a branch. Here you have to make changes.

* git add .

Commit changes

* git commit -m “commit message”

pull from origin branch

* git pull origin <branchname>

pull into remote repository

* git push –set-upstream origin <branchname>

To create a branch

* git branch dev-4.0.0

Checkout into branch

* git checkout dev-4.0.0

Make changes in dev-4.0.0 branch

* git commit -am “commit message”

Push dev-4.0.0 branch into remote repository

* git push –set-upstream origin dev-4.0.0

All the developers develop the code in dev-4.0.0 branch. After completing the development we intimate to QA.

QA pull the code from dev-4.0.0 and then merge into qa-4.0.0 branch. Here they test the code.

Steps for QA:

* git checkout –track -b dev-4.0.0
* git pull
* git branch prerelease
* git checkout prerelease
* git merge dev-4.0.0

Now the qa-4.0.0 branch updated with dev-4.0.0 branch, then QA start testing

Once testing is completed, the code ready to push into master branch.

Steps to merge into master-4.0.0 branch:

* git checkout qa-4.0.0
* git pull origin qa-4.0.0
* git checkout master-4.0.0
* git merge qa-4.0.0
* git push

code is updated in master-4.0.0. If we want to tag here we can tag.

Steps for tag:

* git tag <tag name> (To create tag)
* git push –tags (Push the tags)

Till now we completed one cycle of development.

We release the version to the customer, after few days he find a bug in released version.

Now we have to fix that problem, to fix this problem we have to take code from released version, why because branches are updated with new code. Here we have to create new branch from tag to fix this problem.

* git checkout -b dev-3.5.0 <tag name>

Now the developer fix the problem in dev-3.5.0

After fixing the problem, we merge the code into qa-3.5.0

QA will test the qa-3.5.0 branch and then push the code into master3.5.0 as well master-4.0.0

**Git Commands for Reference:**

**Remote Repositories**

Create a new **connection to a remote repository** (give it a name to serve as a shortcut to the URL):

git remote add [name] [URL]

**Fetch a branch** from a remote repository:

git fetch [remote\_repo] [branch]

**Fetch a repository** and merge it with the local copy:

git pull [remote\_repo]

**Push a branch** to a remote repository with all its commits and objects:

git push [remote\_repo] [branch]

Show the **state of the current directory** (list staged, unstaged, and untracked files):

git status

List the **commit history** of the current branch:

git log

List **all commits from all branches**:

git log --all

**Compare two branches** by showing which commits from the first branch are missing from the second branch:

git log [branch1]..[branch2]

Examine the difference between the **working directory and the index**:

git diff

Explore the difference between the **last commit and the index**:

get diff --cached

See the difference between the **last commit and the working directory**:

get diff HEAD

Display the**content and metadata** of an object (blob, tree, tag or commit):

git show [object]

**Git Branches**

List **all branches** in the repository:

git branch

List all **remote branches**:

git branch -aa

**Create a new branch** under a specified name:

git branch [branch]

**Switch to a branch** under a specified name (if it doesn’t exist, a new one will be created):

git checkout [branch]

**Delete** a local branch:

git branch -d [branch]

Rename branch that you are currently working in:

git branch -m [new\_branch\_name]

**Merge** the specified branch with the current branch:

git merge [branch]

**Making Changes**

**Stage changes** for the next commit:

git add [file/directory]

**Stage everything** in the directory for an initial commit:

git add .

**Commit staged snapshots** in the version history with a descriptive message included in the command:

git commit -m "[descriptive\_message]"

**Undoing Changes**

**Undo changes** in a file or directory and create a new commit with the git revert.

git revert [file/directory]

**Unstage a file** without overwriting changes:

git reset [file]

Undo any changes introduced **after the specified commit**:

git reset [commit]

**Show untracked files** which will be removed when you run **git clean** (do a dry run):

git clean -n

**Remove** untracked files:

git clean -f

**Rewriting History**

**Replace the last commit** with a combination of the staged changes and the last commit combined:

git commit --amend

**Rebase the current branch** with the specified base (it can be a branch name, tag, reference to a HEAD, or a commit ID):

git rebase [base]

List **changes made to the HEAD** of the local repository:

git reflog

**Remote Repositories**

Create a new **connection to a remote repository** (give it a name to serve as a shortcut to the URL):

git remote add [name] [URL]

**Fetch a branch** from a remote repository:

git fetch [remote\_repo] [branch]

**Fetch a repository** and merge it with the local copy:

git pull [remote\_repo]

**Push a branch** to a remote repository with all its commits and objects:

git push [remote\_repo] [branch]