Git Branches

Branches allow us to work in different version of the same file(s) in parallel. Edits, changes on one branch are independent from changes in other branches. We can have branches for different purposes, like production branch, development branch, branch for bug fixes etc. Branch operations:

1. Create branches -git branch <branch\_name>

2. Merge branches -git merge <branch\_name>

3. Delete Branches -git branch -d <branch\_name>

4. Create branch -git branch <branch\_name>

Master : this branch is used for production. We do not directly push into this branch. In companies this branch is protected. Master is the chief branch, main branch.

Develop : this branch is created from master, this is also known as integration branch. Developers make their changes here.

-git branch See all branches The branch name with the star is our current branch

-git checkout develop checks out the branch, switches to the branch

git checkout -b <branch\_name> creates a new branch and switches to it.

-git merge <branch\_name> this command takes changes from the given branch, and merges with the current branches we are on.

We make a change in one branch, and we want carry that change to another branch. In order two merge two branches, we need to

1. Switch to the branch where we want to take the changes to.

git checkout master

2. Run the merge command by mentioning the branch name where the changes are.

git merge develop

Takes the changes from develop and merges into my current branch (master)

git branch -d <branch\_name> deletes the branch.

If we have unmerged changes, this command gives a warning and does not delete.

git branch -D <branch\_name> deletes the branch even if it has unmerged changes. Gives no warning.

git push --set-upstream origin <branch\_name> it creates a new branch with given name in the remote and pushes the changes from local branch to the new branch.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

What is the branching policy/strategy in your company?

What is the branching policy/strategy in your automation framework?

1. We have 2 test engineers in my projects. We have 2 branches, master and develop. We worked on the develop branch. Since we had only 2 people we decided not to use more branches unless we did some big changes. Every sprint, we select tests which have business value and stable, and added them to regression. After that we merged to master.

My daily automated smoke from Jenkins runs against the master branch.

Master branch is stable since we only merge to with once a sprint.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

2. Our test framework is separate repo from the application code repo. Automation framework has a smaller code base, and fewer people are involved. so we do not have a very complicated branching strategy

We have master and develop branches(buffer branch) in our automation framework repository. Once I have a story from jira, I create a new branch from the develop branch with the issue number of my jira story and check out.

git branch <branchName>

git checkout -b vyt-59I

write my automated tests on this branch. Once completed, I create a pull request so that my code can be reviewed. Once my team reviews the code, my branch is merged into develop.

After code is merged, I delete the branch.

$ git branch -d <branch\_name>

Then get another story from jira, create new branch for that one ...

At the end of every sprint, (or quarter) we merge with master.

My daily automated smoke from Jenkins runs against the master branch.

Master branch is stable since we only merge to with once a sprint.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Conflicts generally arise when two people have changed the same lines in a file, or if one developer deleted a file while another developer was modifying it. In these cases, Git cannot automatically determine what is correct. Conflicts only affect the developer conducting the merge, the rest of the team is unaware of the conflict. Git will mark the file as being conflicted and halt the merging process. It is then the developers' responsibility to resolve the conflict.

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\

merge conflict resolution

Conflicts arises when two people have changed the same lines in a file.

Scenario 1:

git status

git add

git commit -m "message"

git push

!!!!problem conflict so what

git fetch

git merge

...fix the conflict

git add

git commit -m "messega"

git push

Scenario 2:

git stash (puts and keeps my working file in a seperate location)

git pull (the most updated file in remote)

git stash pop (merges)

...fix the conflict

git add

git commit -m "messega"

git push

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++