

Integration Platform Modernization

PRJ0012421

**GIT BRANCHING**

Version 1.0.0

**Table of Contents**

[**​1.​ GIT BRANCHING STRATEGY FOR MULESOFT PROJECT**](#_e263drek69n4) **3**

[​1.1.​](#_xwdpsao4ybb8) About git & branching 3

[​1.2.​ Branching in git](#_veusg8fvpjns) 3

[​1.3.​ Git usage](#_kykit06cnkqa) 4

1.3.1. Git clone , commit , merge & delete branch 6

[**​2.​ MULESOFT GIT BRANCHING DETAILS**](#_txtu405cfdi0) **9**

[​2.1.​ Branch overview](#_eh4pey6e20t2) 9

[​2.2.​ Diagram](#_n0gpg080vfb8) 10

2.3 Git branching rules in brief 11

2.4 reference 11

**GIT BRANCHING STRATEGY FOR MULESOFT PROJECT**

**ABOUT GIT & BRANCHING**

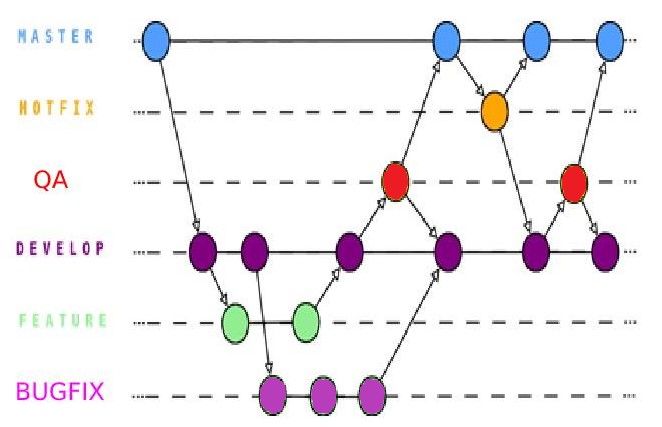
* Git is a free, open source distributed version control system tool.
* It was created by Linus Torvalds in 2005 to develop Linux Kernel



* Git enable communication between the development and the operations team
* In Distributed VCS, every contributor has a local copy or “clone” of the main repository i.e. everyone maintains a local repository of their own which contains all the files and metadata present in the main repository.
* programmer maintains a local repository on its own, which is actually the copy or clone of the central repository on their hard drive. He/She can commit and update their local repository without any interference.
* They can update their local repositories with new data from the central server by an operation called “pull”
* A programmer can update the main repository by an operation called “push” from their local repository.

**Branching in Git**

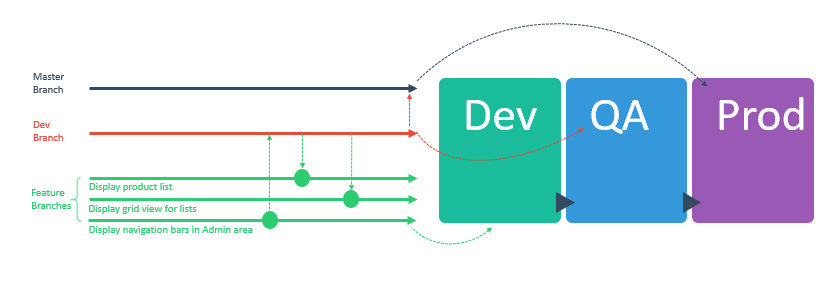
* Branching is a core concept in Git
* A branch in Git is simply a pointer to a commit



* Git was designed to be fully distributed from the start, allowing each developer to have full local control and branches carry their entire history
* When you create a branch in your project, you're creating an environment where you can try out new ideas
* Changes you make on a branch don't affect the master branch, so you're free to experiment and commit changes with in the branch

**Git usage**

Real life projects generally have multiple developers working in parallel. So a version control system like Git is needed to ensure there are no code conflicts between the developers



To start using git , download git.The below link has details on how to install Git in multiple operating systems:

<https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>

How to use git :

* Verify if Git is installed by using the following command in the command prompt:

**git --version**

* To Create a local Git repository:  
  Create a folder named "mulesoft-sample-repo" and initialize git

Steps:

**mkdir mulesoft-sample-repo**

**cd mulesoft-sample-repo**

**git init**

The "git init" command adds a local Git repository to the project.

* Add a file "demo.txt" into the repository with some content in its

Steps:

**Touch demo.txt**

**echo “mule-demo” > demo.txt**

* Staging and Committing the code

Committing is the process in which the code is added to the local repository. Before committing the code, it has to be in the staging area. The staging area is there to keep track of all the files which are to be committed.

Any file which is not added to the staging area will not be committed. This gives the developer control over which files need to be committed.

Stagging command :

Steps:

**git add demo.txt**

Multiple files can be added to the stagging area by adding the file names + path after each space

Steps:

eg: **git add demo.txt file2 src/file3.txt**

If you want to add all the files inside your project folder to the staging area, use the following command

Steps:

**git add .**

* To commit the files tot he local repository, use the following command

Steps:

**git commit -m "** commit message **"**

You can add a message for each commit inside " " above

* To know the current status of the repository

Steps:

**git status**

The above command shows information regarding what files are modified and what files are there in the staging area

* Push the files to the central repository

Steps:

**git push**

After the above command is executed, either a prompt will appear asking for username & password of the central repository account or ssh private key ssh to be configured

Only after successful authentication, the files will be pushed to the central repository

* To see all the commit related details

Steps:

**Git log**

The log shows the author of each commit, the date of the commit, and the commit message.

* To create branches in git

Create a new branch called "test-branch" using the following command:  
Steps:

**git branch test-branch**

Multiple branches are needed to support multiple parallel developments

To create a new branch and switch to it at the same time, you can run the git checkout command with the -b switch

* To switch to the "test-branch" branch. use the following command:

Steps:

**git checkout test-branch**

* To see which branch we are in

Steps:

**git branch**

* To list all the branches inside a repository

Steps:

**git branch –a**

* To merge a file created or updated in "test-branch" branch to master

Steps:

1) **git checkout master**

2) **git merge test-branch**

* In order to point your local repository to the remote repository, use the following command:

Steps:

**git remote add origin [repository url]**

* To pull the latest changes from the remote repository into the local repository

Steps:

**git pull origin master**

This process is done because the remote repository code is updated continuously by various developers, hence git pull is necessary.

**Git clone , commit & merge**

* To clone a remote repository to the local system

Steps:

Git clone **[repository url]**

* To commit a change made in the local repository to remote repository

Steps:

**git add [filepath/file name]** or **git add .** (to add all the updated files in the local repo)

**git commit –m “[message]”**

**git push**

Enter the Nisource username & password when prompted

* To merge a feature branch to dev branch

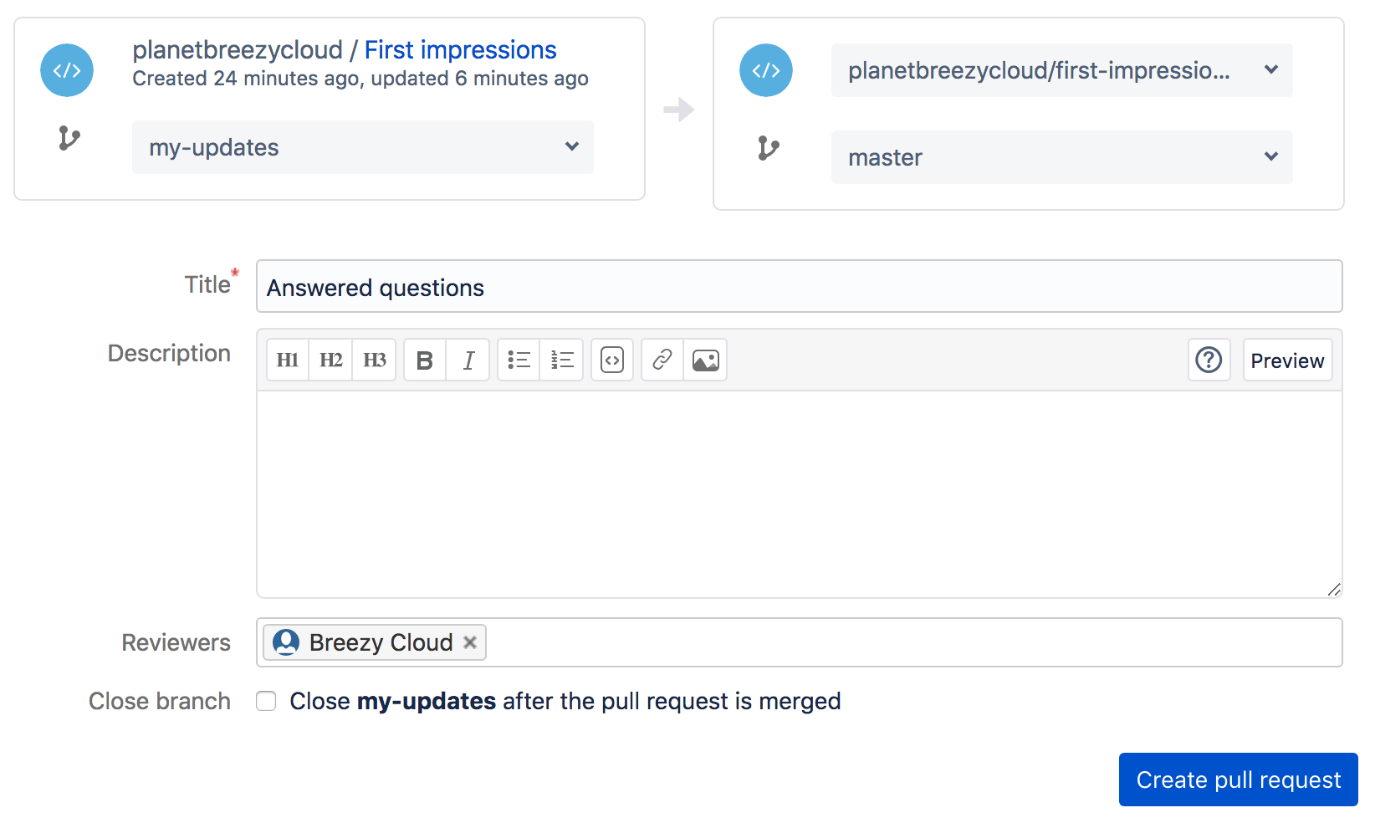
In Nisource , to merge a feature branch to dev branch, we have to raise a merge request (PR aka pull request)

Steps:

Step 1. Create the pull request

You need a branch to create a pull request. Good thing you created a branch in the previous section of this tutorial.

1. From your repository, click **+** in the global (leftmost) sidebar. Then, click **Create a pull request** under **Get to work.**Bitbucket displays the request form. Make sure that you are in a repository before you click **+**in the global sidebar.
2. Complete the form:
   1. You've already got an automatic **Title**, your last commit message.
   2. Add a **Description** if you'd like.
   3. From the **Reviewers** field, add your team member **breezycloud** as a reviewer.

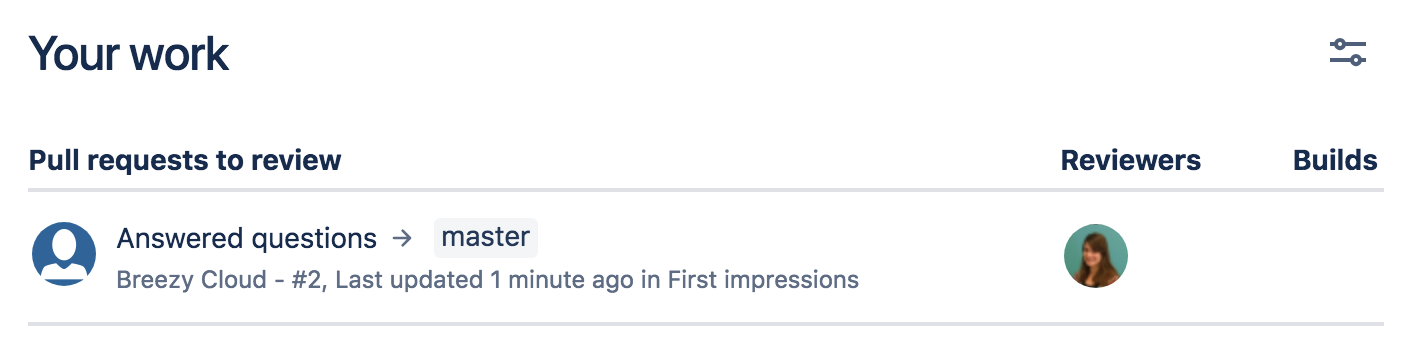
When you're done, the form will look something like this:  


1. Click **Create pull request**.

Bitbucket opens the pull request and your reviewer receives an email notification with details of the pull request for them to review.

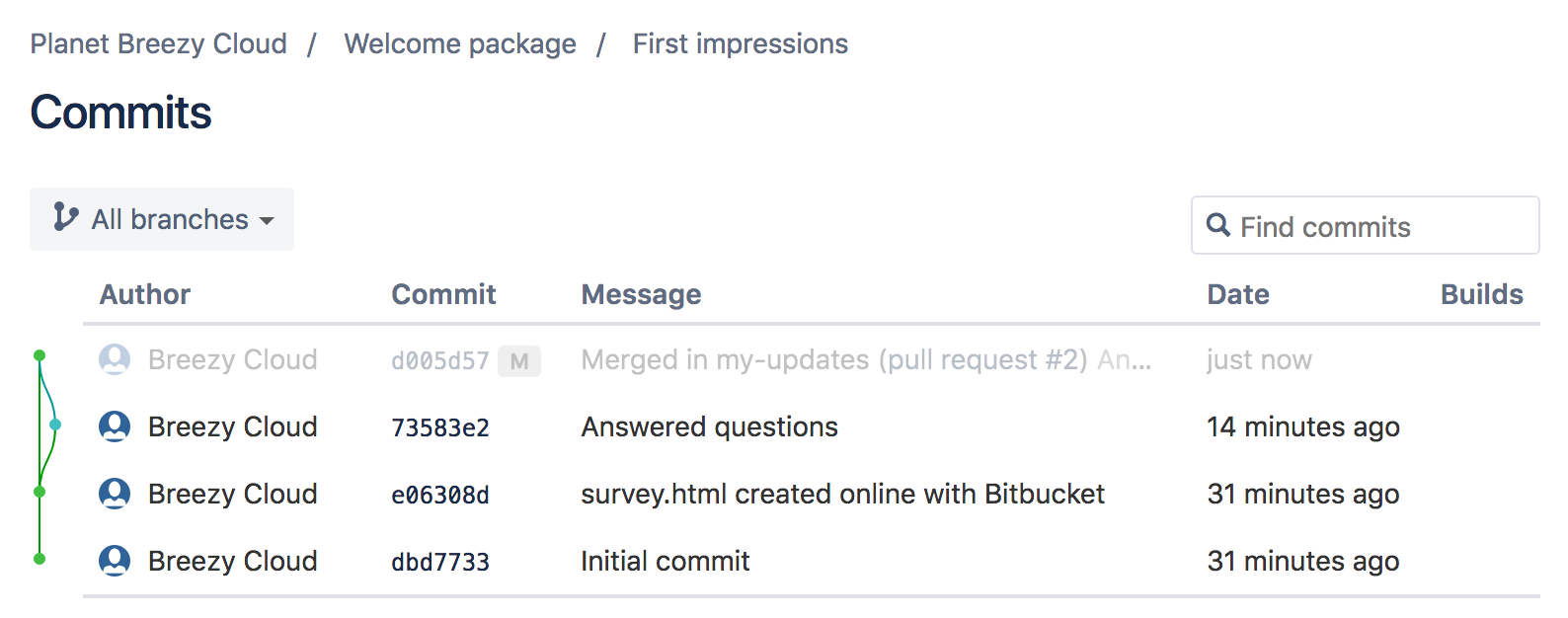
Step 2. Merge your pull request

Not so fast! You may have noticed the **Merge** button at the top. Before you click it, you need to wait for an approval of your changes. In addition to the email notification your teammates receive, they'll also see the pull request appear under **Pull requests to review** on the **Your work** dashboard.



From the pull request, the reviewer can view the diff and add comments to start a discussion before clicking the **Approve** button.

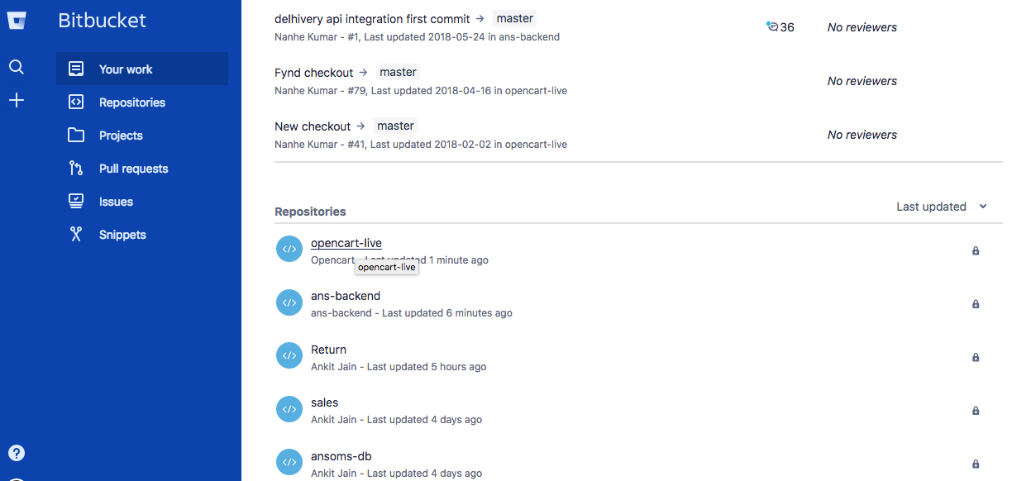
When someone approves your pull request, you'll get an email notification. Once you've got the approvals you need (in this case just one!), you can merge. From the pull request, click **Merge**. And that's it! If you want to see what it looks like when your branch merges with the main branch, click **Commits** to see the commit tree.



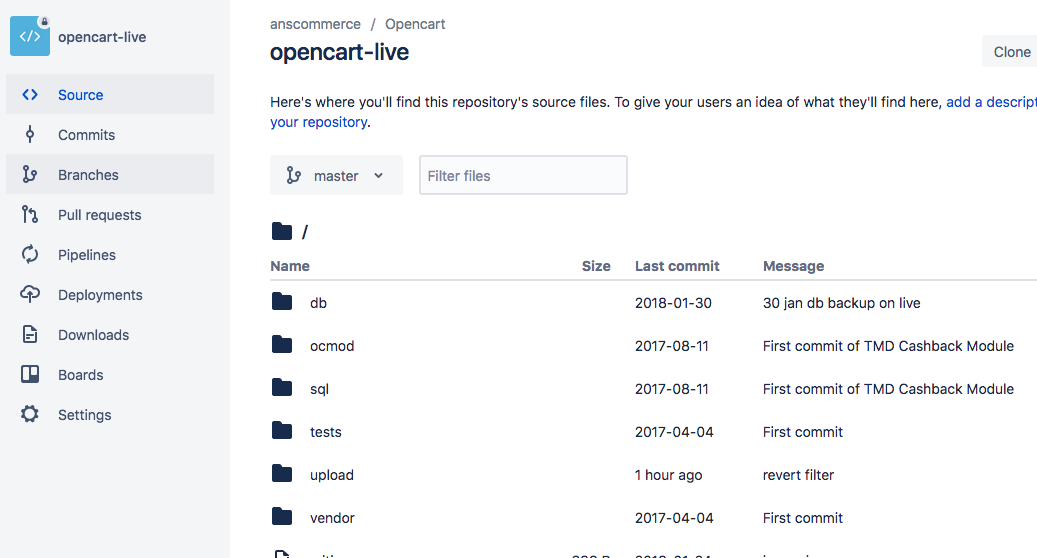
* Delete a branch from a repository

Step 1 : Login in **Bitbucket**

Step 2 : Select Your Repository in Repositories list.

[](https://i.stack.imgur.com/ExwYG.png)

Step 3 : Select branches in left hand side menu.

[](https://i.stack.imgur.com/be5dz.png)

Step4 : Cursor point on branch click on three dots (...) Select Delete (See in Bellow Image)

[](https://i.stack.imgur.com/RNHcc.png)

**Mulesoft Git branching details**

**Overview**

* MASTER
* QA/STAGING
* DEV
* FEATURE
* HOTFIX

**MASTER**  
The master branch at origin should be familiar to every Git user.We consider origin/master to be the main branch where the source code of HEAD always reflects a production-ready state.

**QA/STAGING**  
QA/STAGING branches support preparation of a new production release. All the testing part will occur in this branch. They allow for minor bug fixes and preparing for a release

**DEV**  
Dev branch is called as the “integration branch”. All the Feature branches will be merged backed to Dev branch Feature branches are created from Dev branch.

When the source code in the develop branch reaches a stable point and is ready to be released, all of the changes should be merged back into master somehow and then tagged with a release number.

**FEATURE**  
Feature branches are used to develop new features for the future or new release.

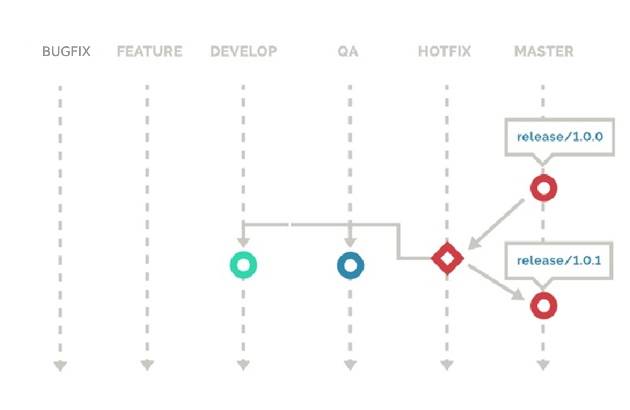
**BUGFIX**Typically used to fix Master & QA branches.

**HOTFIX**  
When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version.

Hotfix branches are created from the master branch.

When finished, the bugfix needs to be merged back into master, but also needs to be merged back into develop

GIT BRANCH DIAGRAM



**Git branching rules in brief**

|  |  |
| --- | --- |
| SL | BRANCHING RULES AND DETAILS |
| 1 | MASTER” and “QA” branches should have the same code |
| 2 | “FEATURE/##” branches are to be created from “DEV” branch |
| 3 | After development in “FEATURE” branch, the code has to be merged to “DEV” branch |
| 4 | With respect to “MASTER” and “QA” branches, the code in Dev branch may differ according to the feature branch created |
| 5 | After development & unit testing is completed for a particular functionality, the code has to be merged to “DEV” branch and the related “FEATURE” branch has to be deleted |
| 6 | After basic testing in “DEV” environment is complete, the code has to be merged to “QA/STAGING” branch and deployed to the desired environment for regression, load, performance testing. |
| 7 | UAT has to be performed in the “QA/STAGING” environment |
| 8 | After all testing(sign-off from tester) and UAT process has passed, the code has to be merged with “MASTER” branch |
| 9 | After PROD deployment is done, confirm that “MASTER” , “QA/STAGING” , “DEV” branches in git repo have the same code and feature branches are deleted |
| 10 | Bug fix has to be performed by creating the a “FEATURE/##” branch from “DEV” branch |
| 11 | Hot fix has to be performed by creating the a “HOTFIX/##” branch from “MASTER” branch |
| 12 | After “HOTFIX” is done, it has to be merged with “MASTER” , “QA/STAGING” & “DEV” branch |

Note: “##” means topic name or functionality name or developer name

**Reference**

For more information refer:

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