Jenkins Pipelines

Jenkins Pipelines

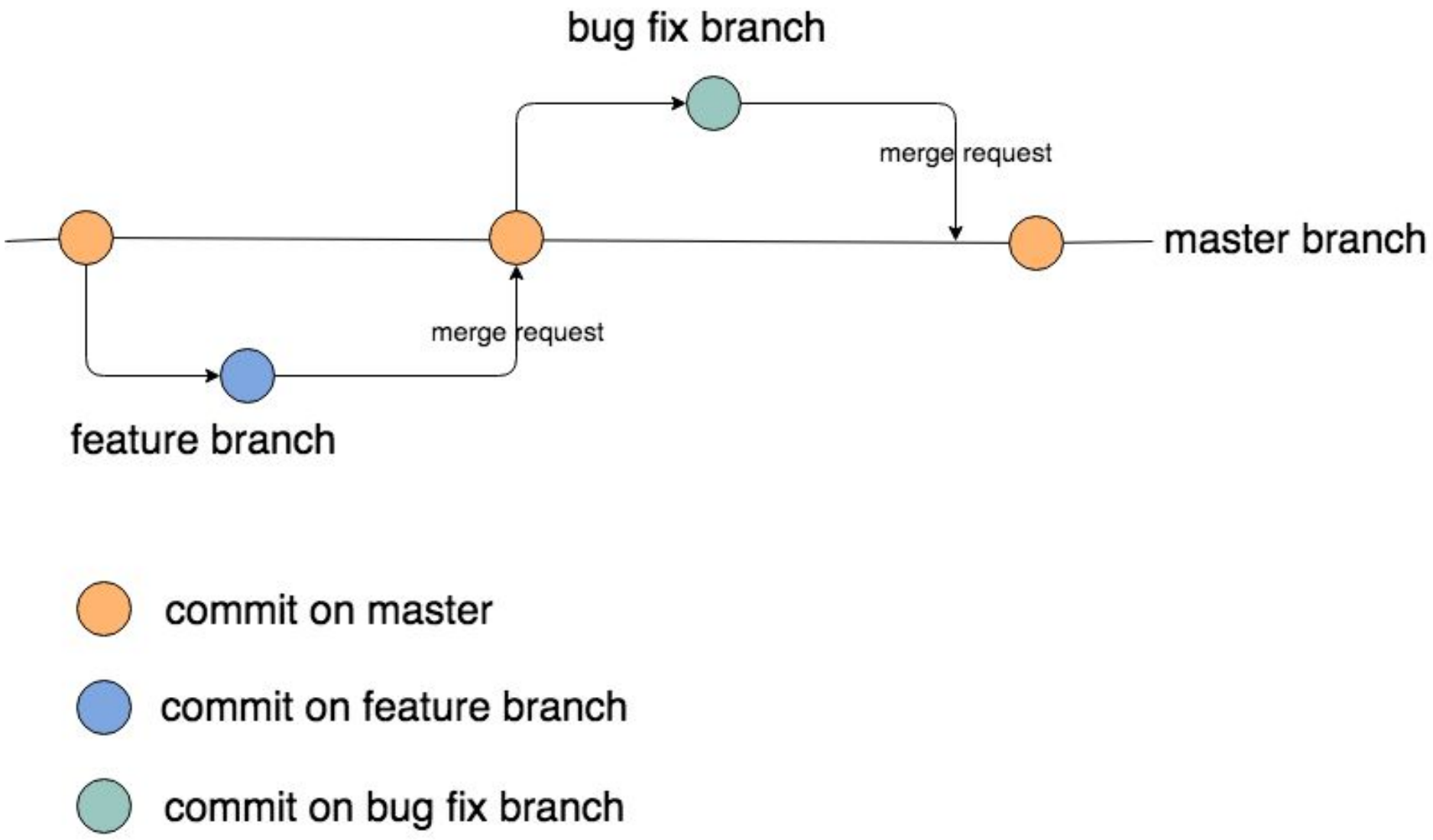
- Identify Git workflows that enable CI and easily integrate into Jenkins
- Use a version-controlled project with multiple branches and build it on Jenkins
- Use the declarative Jenkins pipeline and add pipeline to version control

The CI Workflow

CI Pipeline Steps

- Pulling Code from Source Control
- Preparing the Application Environment
- Testing
- Building
- Deployment

Git Branches

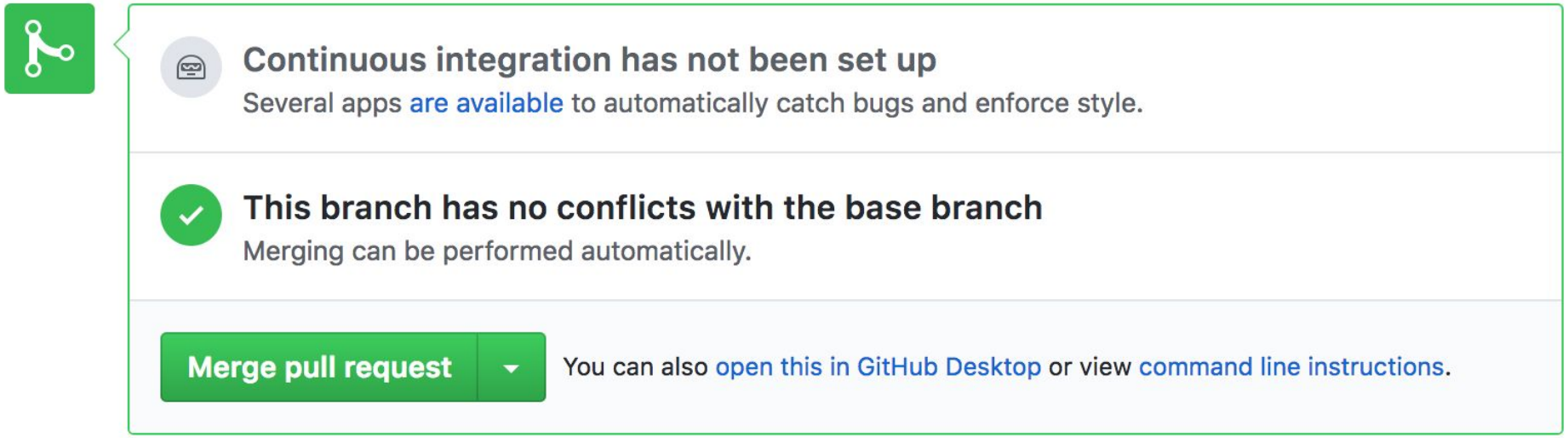


Setting up our Repository

- Create a New Github Repo named – IBM_JenkinsPipelines
- Clone Github repo to our local repository
- Create a new branch called **add-functions-and-tests**.
  - git checkout -b add-functions-and-tests
- Create the files in the root folder of the project and push to Remote
  - git commit -m "add functions plus unit tests"
  - git push origin add-functions-and-tests
- Going back to our project dashboard on GitHub, we can see that a new branch has been added

Setting up our Repository

- Let's create a pull request to our master branch, allowing us to merge the changes from our new branch.
  - Select the **Compare & pull request** button to create and configure our pull request



Creating a GitHub Repository & Integrating Jenkins

- To create a pipeline project by integrating Jenkins with a GitHub repository, follow these steps:
  - Go to the GitHub dashboard and create a new repository.
  - On the repository configuration page, initialize the project with a README file.
  - Clone remote repo locally
  - Checkout to a new branch called add-code-files. Add the provided code samples to your project while under this branch and push the changes to the remote repository.
  - Create a pull request from your new branch to the base branch of master.
  - Go to the repository settings and add webhook
  - For the Payload URL **http://your-jenkins-url/github-webhook/**

The Jenkinsfile

The Jenkinsfile

- A pipeline in Jenkins is defined using a script called the **Jenkinsfile**
- While working with the Jenkins scripted pipeline, we use standard Groovy syntax
- The scripted pipeline has some special directives that perform different functions

The Jenkinsfile

Directive	Explanation node
`node`	This defines where the job is going to be run. We will explore more about this in the next chapter as we cover setting up master-slave relationships on Jenkins.
`dir`	This directive defines what directory/folder to run the following directives on.
`stage`	This defines the stage of your pipeline, for example, what task it's running.
`git`	This points to the remote repository where you pull the changes from.
`sh`	This defines the shell script to run on a UNIX-based environment. On a Windows environment, we would use the `bat` directive instead.
`def`	As mentioned previously, the pipeline is written in Groovy; thus, we can define functions to perform different actions. In this case, we defined a `printMessage` function, which prints out different messages at the start and end of our pipeline.

Creating the Pipeline

- Go to the Jenkins dashboard and select New Item.
- Enter an appropriate name(PipeLine-Project-1) for the project and select Pipeline for the project type
- In the project configuration, under the **General** tab, select **GitHub project** and enter the appropriate URL
  - https://github.com/atingupta2005/Jenkins-5-Days-Training-Material
- Under the **Build Triggers** section, select the **GitHub hook trigger for GITScm polling**
  - Need to create a Webhoob in Github Repo – Settings->Webhook
    - http://52.142.55.134:8080/github-webhook
- Under the **Pipeline** section, select **Pipeline script** under **Definition**.
- In the script section of the configuration, add the snippet of code:

Creating the Pipeline

```
node('master') {
    stage("Fetch Source Code") {
        git 'https://github.com/atingupta2005/Jenkins-5-Days-Training-Material.git'
    }
    dir('Hands-On/Participants/7. Jenkins in Action/8. Jenkins Pipelines') {
        printMessage('Running Pipeline')
        stage("Testing") {
            sh 'python test_functions.py'
        }
        printMessage('Pipeline Complete')
    }
}
def printMessage(message) {
    echo "${message}"
}
```

Creating the Pipeline

- Press Apply
- Select Save
- Select Build Now
- On the project dashboard, after running our build, the Stage View shows up.

Installing Blue Ocean

- Jenkins has a new way of visualizing pipelines, called Blue Ocean.
  - On the Jenkins home dashboard, go to Manage Jenkins -> Manage Plugins
  - Under the Available tab, search for Blue Ocean.
  - Select Blue Ocean and click Install
  - New item has been added to our configuration panel on the left, called Open Blue Ocean
  - The Blue Ocean dashboard will display your project on your Jenkins server

Creating Multibranch Pipelines

Creating Multibranch Pipelines

- Multibranch pipelines will enable you to build different branches besides the default.

Global Variables

- A global variable is accessible in any scope within our program
- There are pre-defined global variables. Examples:
  - BRANCH_NAME
  - BUILD_NUMBER
  - BUILD_ID
  - JOB_NAME
  - NODE_NAME
  - JENKINS_HOME
  - BUILD_URL

Create Multibranch project

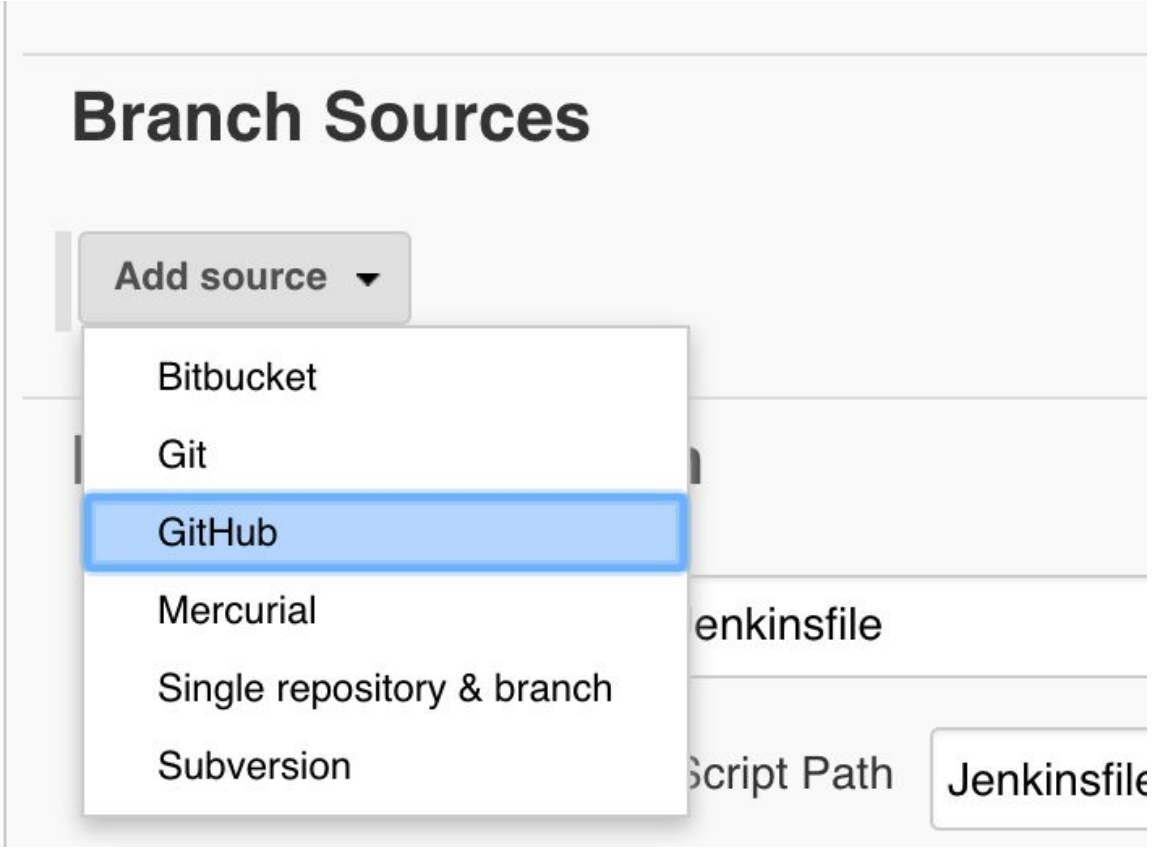
- Add the following updated script to pipeline to create a new multibranch project
- Add Jenkinsfile to version control
  - Add this script to the staging area, create a commit, and push it to the GitHub repository.

Jenkinsfile

```
node('master') {
  stage("Fetch Source Code") { git 'https://github.com/atingupta2005/Multibranch-Pipeline.git' }
   dir('') {
     printMessage('Running Pipeline')
     stage("Testing") { sh 'python test_functions.py' }
     stage("Deployment") {
        if (env.BRANCH_NAME == 'master') { printMessage('Deploying the master branch') }
                  else { printMessage("No deployment for branch [${env.BRANCH_NAME}]") }
     }
     printMessage('Pipeline Complete')
  }
}
def printMessage(message) { echo "${message}" }
```

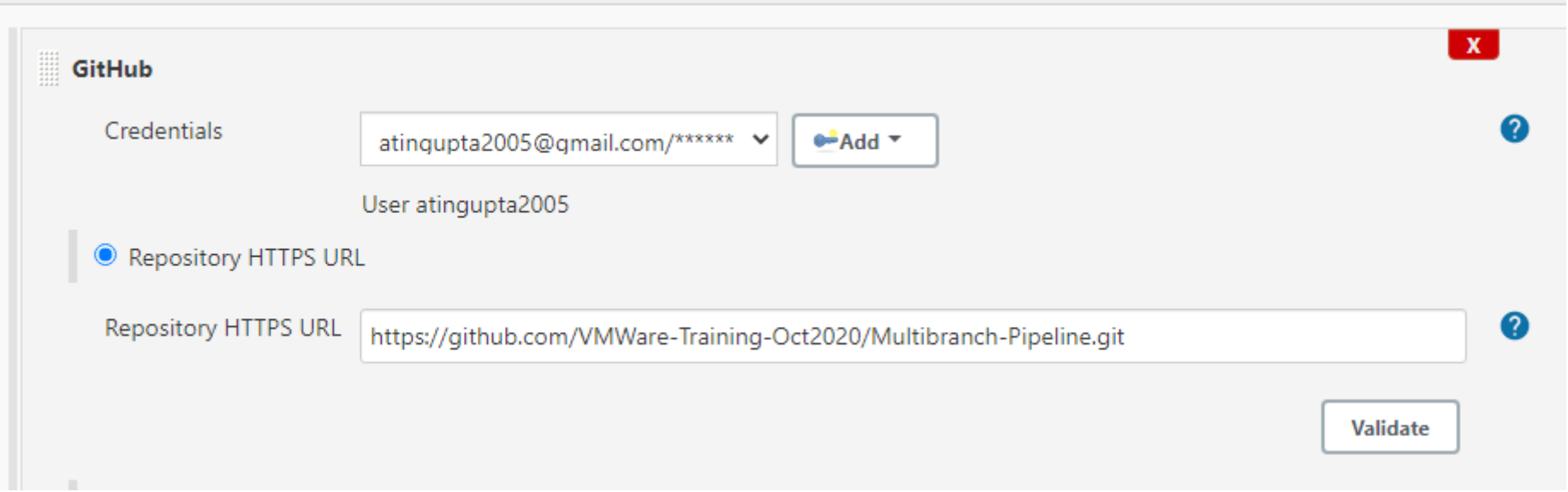
Creating Multibranch Pipelines

- Create a new project on Jenkins and, under the Project type, select Multibranch pipeline and enter an appropriate name for the project.
- Under Branch Sources, select Add source and select GitHub
- Under Kind, select Username with password.

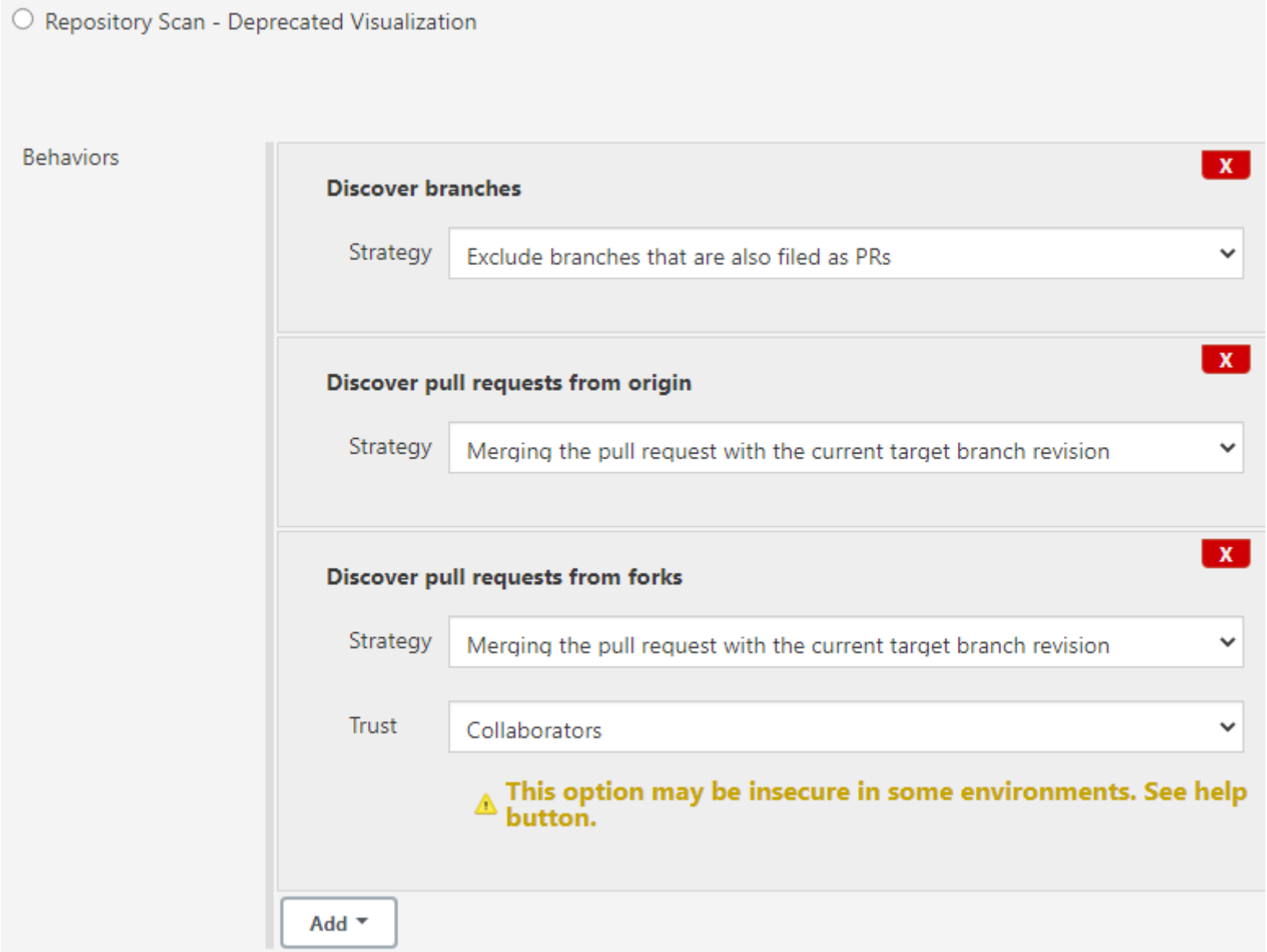


Creating Multibranch Pipelines

- Enter your GitHub username and password in their respective fields and select Add
- Back on the project configuration page, select the credentials you just created under the Credentials section.
- Under the Build Configuration section, set up the project as shown.



Creating Multibranch Pipelines



- Select Apply and Save
- Open Blue Ocean
- Under the Branches tab, select the first item, which in our case is master

Building Pull Requests

- On your Git Bash in the project root, check out to a new branch and do some modifications in the code
  - https://github.com/atingupta2005/Multibranch-Pipeline.git

- Push the new branch to the remote and create a new pull request.

- Going back to Jenkins, we can see our new branch and, under the Pull Requests tab
- We can see that the Pull Request we created has been built by Jenkins using the same pipeline stages.

Thanks