DEVELOPMENT PROCESS

1. **GitHub Account**

To access repositories on GitHub, developer needs to submit a request for IT to create a GitHub account. This account will be the same as the domain.

**GitHub Training**

If you are not familiar with Git, you can sign up for online training classes.

Online training: <https://services.github.com/training/>

1. **Git CLI**

Git command line interface (CLI) can be installed from <https://git-scm.com/downloads>

1. **Git Configuration**

Git comes with a tools called *git config* that let you get and set configuration variables that control all aspects of how Git looks and operates. Following are important variables to set when working with Git:

$ git config –global user.name <yourGitUserName>

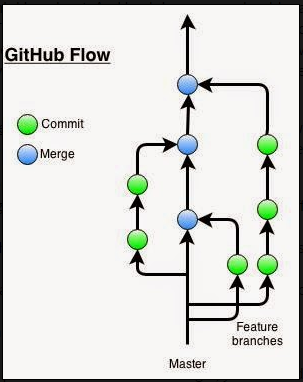
$ git config –global user.email <yourEmailAddress>

$ git config –global push.default simple

“push.default simple” will *push* only the current branch to the one that *git pull* would pull from and also checks that their names match. This is to avoid pushing all local branches accidentally.

1. **GitHub Flow:** GitHub Flow is based on a set of simple business choices:

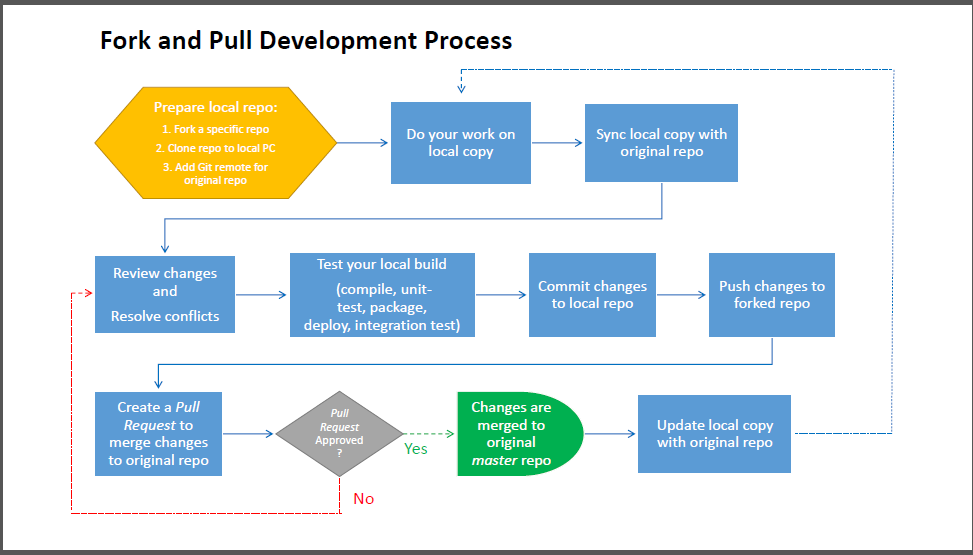
* Anything in the *master* branch is deployable
* To work on something new, create a descriptively named branch off of master
* Commit to that branch locally and regularly push your work to the same named branch on the server
* When you need feedback or help, or you think the branch is ready for merging, open a *pull request*
* After reviewer/approver has reviewed and signed off on the feature, approver will merge your changes into master repository.
* Once it is merged and pushed to *master* on the origin, you can and should deploy immediately

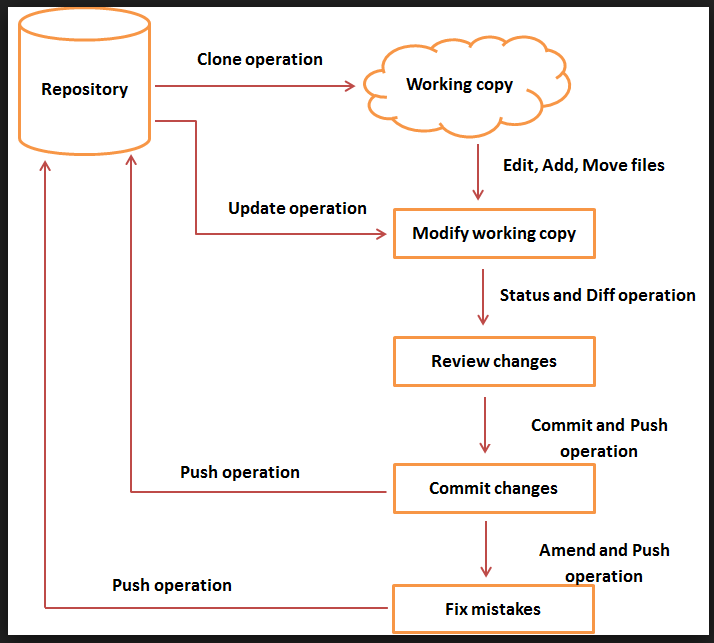


1. **Create a private repository using Fork and Pull Workflow**

Apps developers use fork and pull model for development. The fork and pull model lets anyone fork an existing repository and push changes to their personal fork without requiring access be granted to the source repository. The changes must then be pulled into the source repository by the project maintainer. This model reduces the amount of friction for new contributors and is popular with open source projects because it allows people to work independently without upfront coordination.

The process:

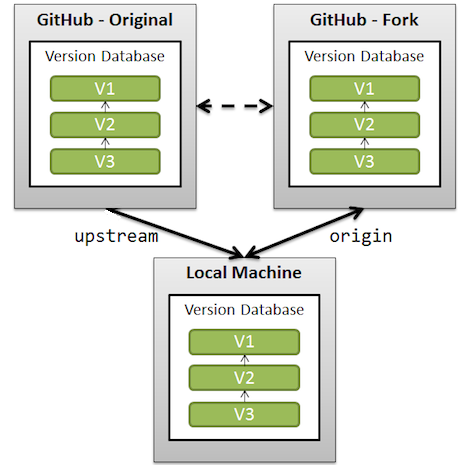




When the repo is cloned, it has a default remote called *origin* that points to your forked repo on GitHub. To keep track of the original repo, you need to add another remote named *upstream*.

You will use *origin* to pull and push since you can contribute to your own repo.

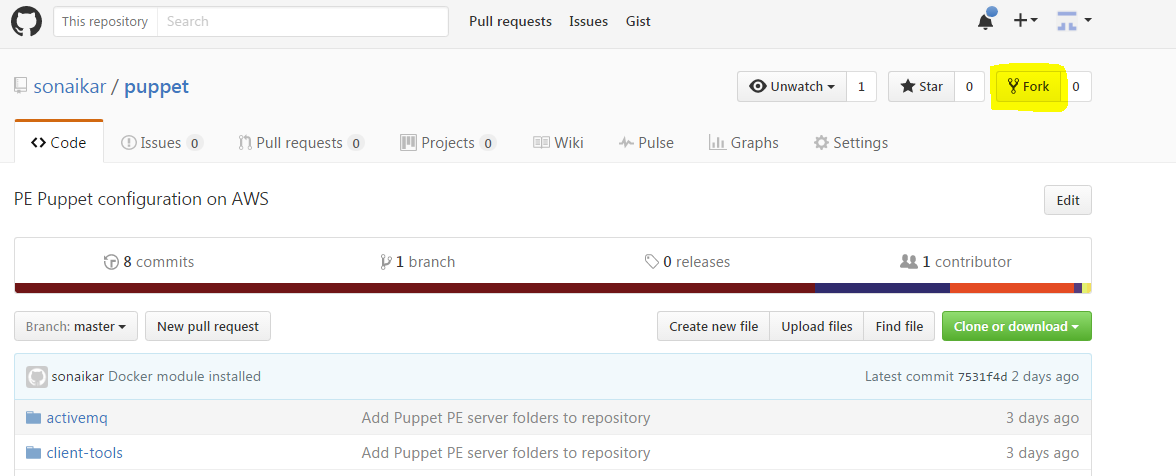
You will use *upstream* to fetch from the original repo in order to keep your local copy in sync with the project you want to contribute to. You will contribute back to the *upstream* repo by making a *Pull Request*.



The steps:

1. **Fork a repository**

To create a private copy of a repository, navigate to the repository and select the  option in GitHub from the top right corner of the screen in as shown below:

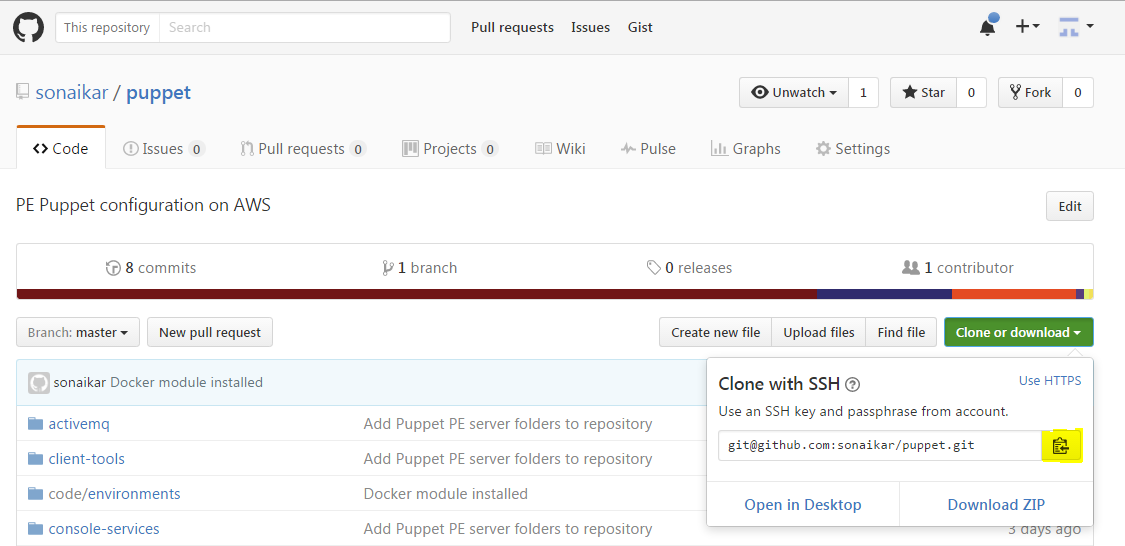


Select your account when the following popup screen appear

It will take a few seconds to complete

1. **Clone your repository**

* Click on Copy to Clipboard to copy the URL of your repository.



* Go to a directory on your computer where you want to clone the repository to and execute the *git clone* command:

**$ git clone** [**<Git**](https://github.bmc.com/mle/hrcm.git) **repository URL>**

1. **Add a Git remote for the original repository**

$ git remote add upstream https://github.rbc.com/<path>/project1.git

1. **Do your works**

Do your development work in workspace, i.e., add/delete/modify/move files.

* + **Check status of your changes**

$ **git status**

1. **Sync your local copy with the original repository**

$ git pull upstream master

1. **Review changes and resolve conflicts if any**

If Git could not automatically resolve conflicts, all conflicts will be marked using the following scheme:

<<<<<<< HEAD:file.txt

This what developer has locally in file.txt

=======

Here is the changes were received from the remote repository

>>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086

Developer should manually fix all conflicts in local files and then commit the files as usual.

1. **Test your local build**
   * Build your project:

$ cd <yourProjectFolder>

$ mvn clean install

* + Run unit tests:

$ cd <yourProjectFolder>

$ execute test cases

1. **Commit your changes to local repository**
   * **Add the changes you make to local repo**

$ git add .

This command add all files (modified and new including the one whose name begin with a dot) in current directory to the staging area but does not remove files that have been deleted from the disk.

**$ git add \***

**This command is the same as “git add .” but exclude files whose name begin with a dot**.

*Warning: LF will be replaced by CRLF message can be ignored*

warning: LF will be replaced by CRLF in tools/abc.java.

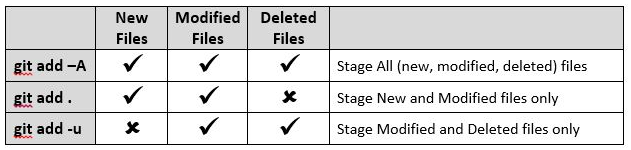
The file will have its original line endings in your working directory.

$ git add –u

This command add all modified and deleted files to the stage area.

$ git add –A

This command add all files (modified, new, and deleted) to the stage area.



* + **Commit your change**

$ **git commit** –m “Commit details and comments to describe your commit”

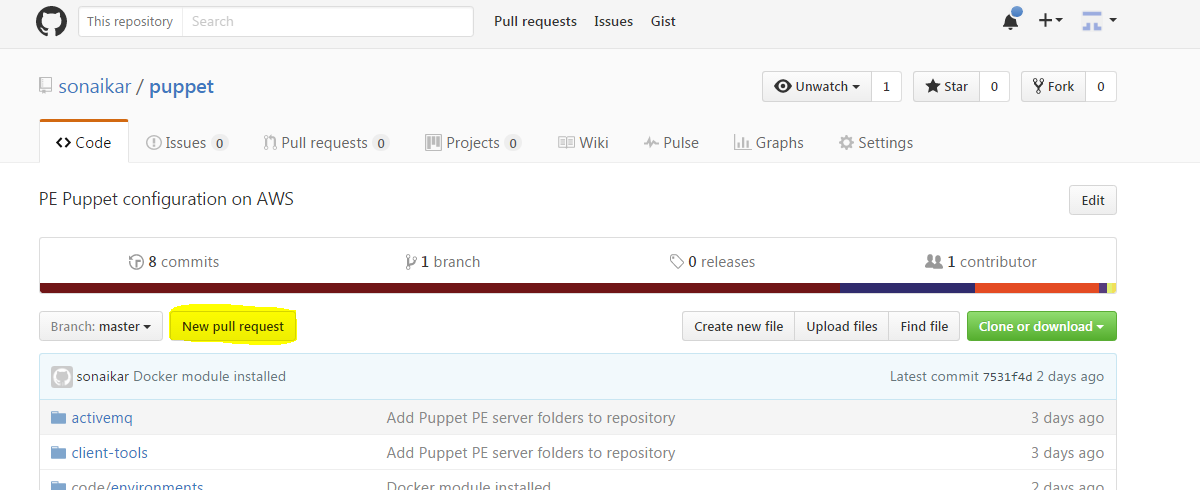
1. **Push your changes to the forked repository**

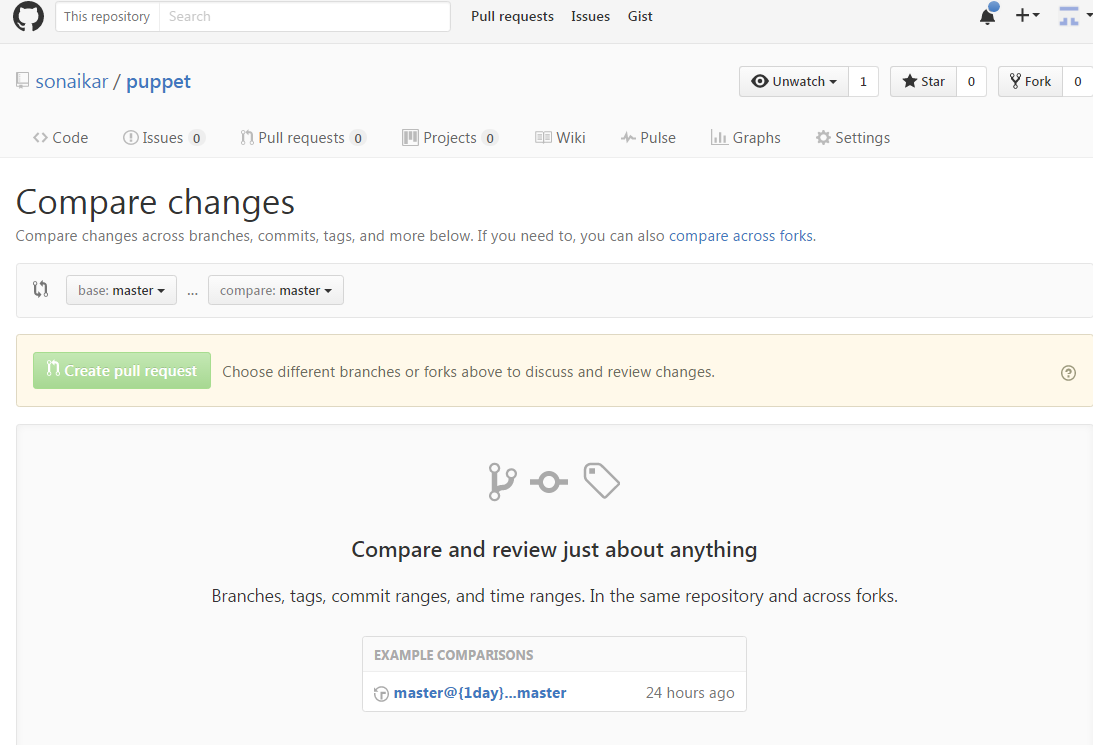
$ **git push** origin master

1. **Create a Pull Request to merge your changes to the original repository**
   * Open the forked repository
   * Select the  tab
   * Click on the  button.
   * Click on the  button.
   * Describe the pull request
   * Click on  button

The project maintainers will receive email notification of the *Pull Request.* Project maintainers will review and approve or comment on the request if there are some changes need to be done. Once the *Pull Request* is approved, the project maintainer will merge the changes in the *Pull Request* to the original master repository. Developer will receive email notification once *Pull Request* is approved.

Refer to the following screenshots for an example:

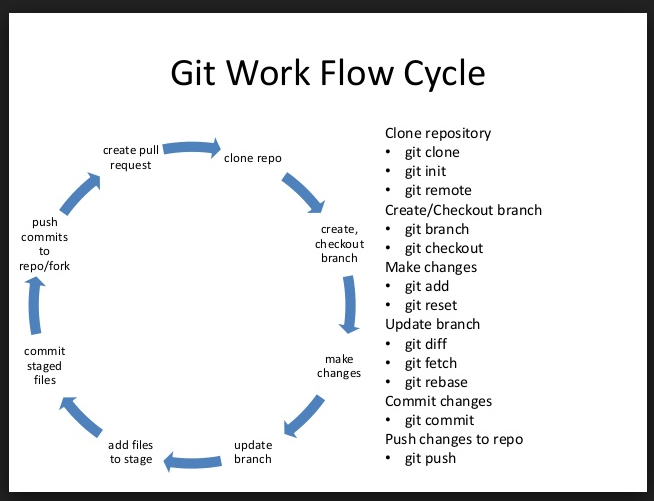




1. **Update your local repository from the original repository**

$ git pull upstream master

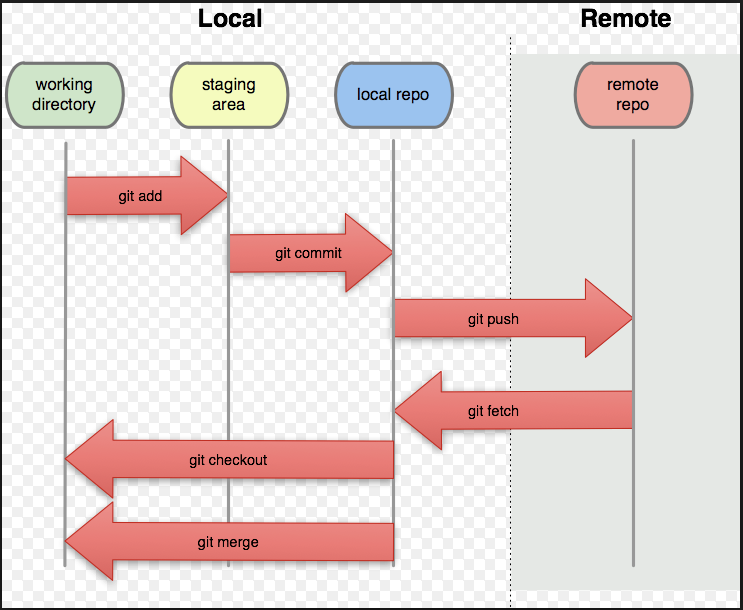
1. **Git Work Flow Cycle**



1. **Git change commit process:**

There is a 3 step process:

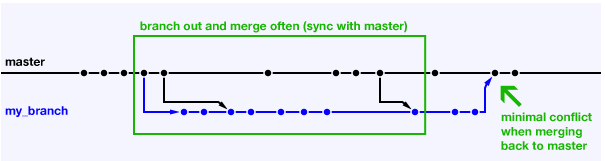
* 1. Stage the change (git add)
  2. Commit the change (git commit, update local metadata)
  3. Push the change (push changes to remote repository)



1. **Follow the good practice: Branch out, Merge often, and Keep in sync**

* Never check in anything directly to the *master* branch

After branching out, you should keep in-sync with the ‘*Development or Master’* branch because eventually you need to merge it back to master. In order to avoid a huge complicated conflicts when merging back, you should commit often, merge often. A good practice would be pulling the changes from *master* into your branch at the beginning of each work day and solve conflicts if any daily. This will help merging back to the *master* branch smoothly.



1. **Project Maintainer/Reviewer**

Each project should assign project maintainers and reviewers. They can be developers or leads who work on the project, knowledgeable in project development and drive the project direction. Owner of the project can assign someone with this role under the Settings/Collaborator & Teams tab.

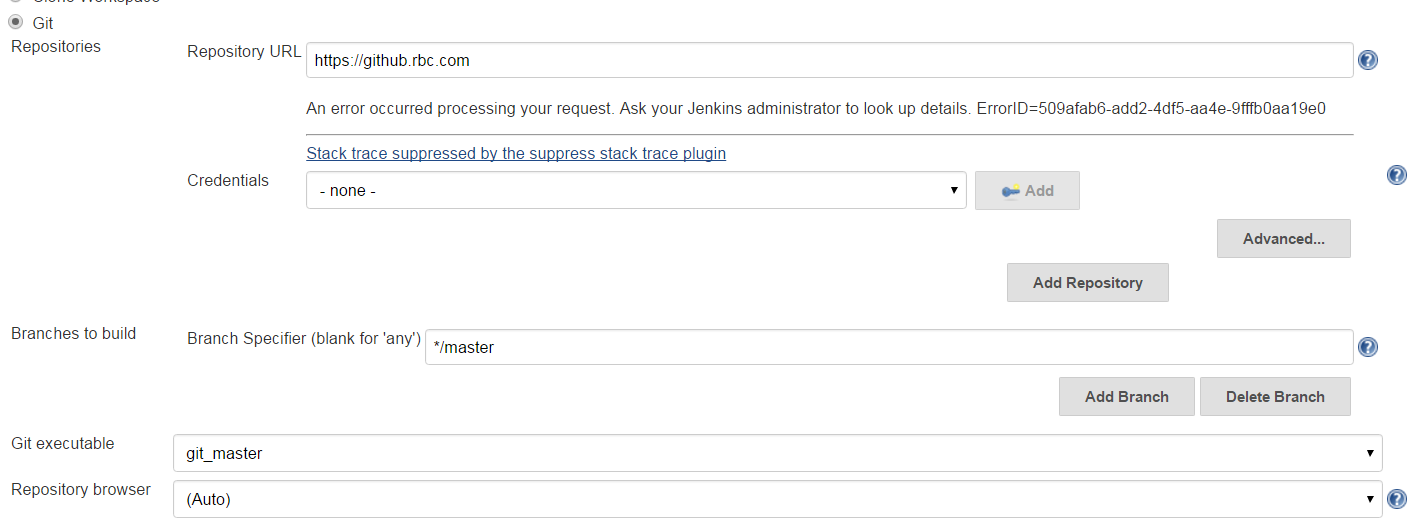
1. **Git Editor**
2. [Atom](https://atom.io/) **:-** [**https://atom.io/**](https://atom.io/)
3. Git Bash :- <https://git-scm.com/downloads>
4. Notepad++

To associate Git with Notepad++ editor, for example, use the following command:

$ git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst -notabbar -nosession -noPlugin"

**II. BUILD PROCESS**

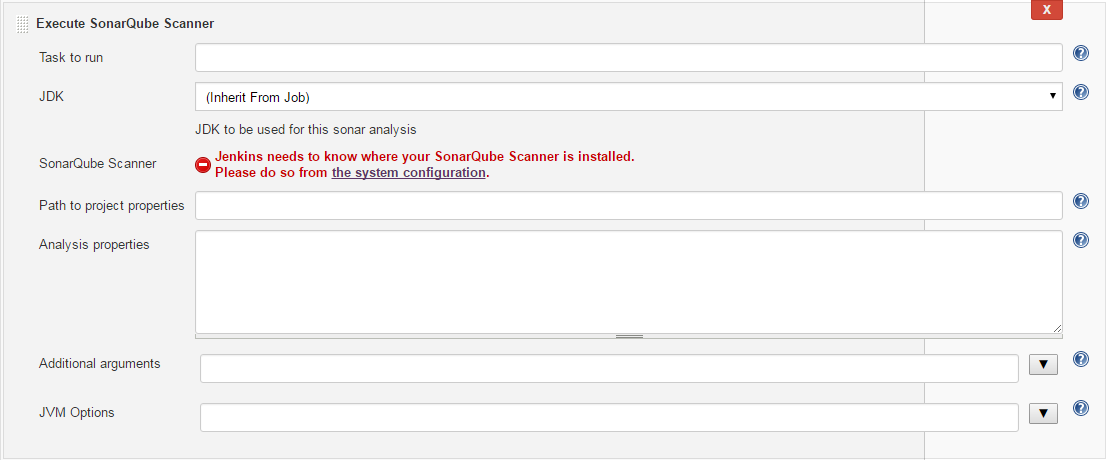
1. Apps nightly build can be configured and run in Jenkins server. Below is snapshot of git repository added to sync code for nightly builds.



1. On-demand build can also be kicked off if you have permission to start build.
2. **Source Code Analysis**

SonarQube, Clover, Klocwork or other tools can be used for analyzing the source code. These tools can be integrated into Jenkins build to check the code quality and collect points for improvement in the projects.

Example of SonarQube scanner configuration.



On the daily or weekly basis, project maintainers should check Static code analysis results and allocate necessary development resources to work in the concern areas.

Preferably, every project should add static code analysis scan step to the Jenkins build