Gerrit Configuration & Code Review Best Practices

Version 1.18-SNAPSHOT

2018-08-22

Table of Contents

1. Server configuration 1
    1.1. Initialization 1
        1.1.1. Permissions 1
        1.1.2. Verified status 2
        1.1.3. Fast Forward 2
    1.2. User preferences 2
    1.3. Project creation 2
        1.3.1. Project git address 2
        1.3.2. Users groups creation 3
2. Code review golden rules (using Gerrit) 3
3. Appendix 6
    3.1. Revision marks 6

Table 1. History

Date	Author	Detail
2018-08-22	NeVraX	HTML Asciidoc to Github

1. Server configuration

Connect to Gerrit homepage.

1.1. Initialization

This has to be done only for a new Production Line

1.1.1. Permissions

Jenkins user push

- Click on **People** → **List Groups** → **Non-interactive Users**
- Add Jenkins (your technical account) in the list

Deleting tags

- Click on **Projects** → **List** → **[All-projects]** → section **Access** → **Edit**
- Under **[Reference: refs/tags/*]**
  - Click on **[Add Permission...]** and select **Push**
  - Select group **Administrator** and click **Force Push**
  - Save Changes

Now you can delete tags from your projects, for ex :

```
git push --force --delete origin cg-wm-1.17.6
```

1.1.2. Verified status

- Click on **Projects** → **List** → **[All-Projects]** → section **General** → **Edit Config**
- Add this

```
[label "Verified"]
   function = MaxWithBlock
   value = -1 Fails
   value =  0 No score
   value = +1 Verified
```

- Click on **Save**, then **Close**
- Click on **Publish Edit**, then **Publish**, **[Code-Review+2]**, **Submit**
- Click on **Projects** → **List** → **[All-Projects]** → **Access** → **Edit**
- Under **[Reference: refs/heads/*]**
  - Click on **[Add Permission...]** and select **Label Verified**
  - Select group **Administrator**
  - Select group **[Non-Interactive Users]**
  - Save Changes

1.1.3. Fast Forward

By default, when projet submissions are not fast forward, final submitting a change will create a merge commit. The history is potentially doubled.

- Click on **Projects** → **List** → **[All-Projects]** → **General**
- Under **Submit Type**, select **Rebase if Necessary**

1.2. User preferences

Click on **YourName** → **Settings** → **Diff Preferences** and set **columns** = **120** (you will probably have to paste it due to a GUI bug)

1.3. Project creation

Create your GIT project by clicking on **Projects** → **Create New Project**

- Project Name = **cg-wm**
- Rights Inherit From = **All-Projects**
- Check that it has inherited correctly "Rebase if necessary", else change and save

1.3.1. Project git address

The git is visible in **Projects** → **[(gitweb)]**.

Something like :

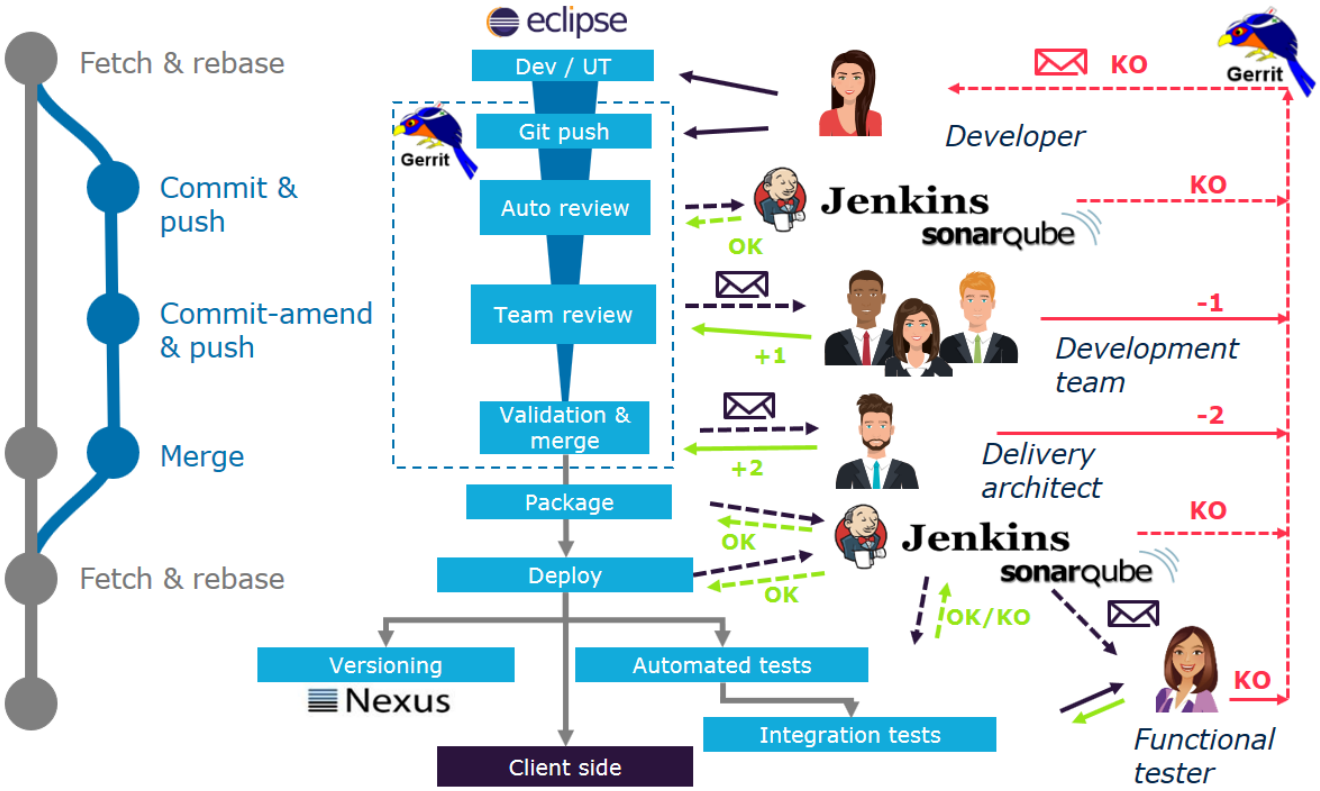
- https://cric.pl.s2-eu.nvx.com/gerrit/cg-wm.git

1.3.2. Users groups creation

For each project, create a reviewer list and a validator list.

- Go to **Projects → Create New Group**
- Reviewers list
  - Give a name, for example **[dge-reviewers]**
  - Add every developers / primary reviewers on the project
  - Click on **General**
  - Description = Reviewers (first level : +1)
  - Click **Save Description**
  - Check **[Make group visible to all registered users.]**
  - Click **Save Group Options**
- Validators list
  - Give a name for example **[dge-validators]**
  - Add technical responsible and a backup
  - Click on **General**
  - Description = "Validators (level 2 : +2)"
  - Click **Save Description**
  - check **[Make group visible to all registered users.]**
  - Click **Save Group Options**

2. Code review golden rules (using Gerrit)



These are rules to be followed for a smooth overall development process :

1. The team has to know that you are taking responsability of the current development task.
2. If not sure of what to achieve, confirm with task responsible
3. Target a complete realization in delays estimated by team leader. Alert on time shortage.
4. Update documentation along with code whenever it's needed.
5. Do not group functionnalities in commit, to avoid long run reviews.
    a. It is possible to handle multiple review in parallel.
6. Commit text has to be explicit, complete, and synthetic.
    a. Commit text must be one line for the sake of history and documentation readability (replace ':' with '()' and '-' with '+'). No limit to the lengh of the line.
    b. If the commit include documentation, set a first line commit text suitable for documentation. Put other information on other lines (they won't appear in documentation history)
7. Commit often, at least on tuesdays and thursdays (even on unfinished current task).
8. No "related changes" should appear on the change in Gerrit, or you did not handle multiple review properly.
9. Fixing Jenkins failures is always a top priority.
10. On "Cannot merge" Gerrit message, you have to pull/commit/push to rebase properly
11. When Jenkins give a +1, add the **reviewers** list as reviewers, this should add all reviewing people.
12. When added as a reviewer, try to give a review in the next half day, knowing that it blocks the process.

a. You don’t have to be an expert to do a review. At least try to spot pieces of code not well explained and missing javadoc. Try to imagine yourself as a future bug fixer who needs clean code to work properly.

b. If suitable, test the application or check the auto IT tests.

c. If any, check that the generated documention looks good in PDF.

d. Check that there is UT specifically testing the new/modified code.

13. When one or two +1 from the team have been given (depending on the size of the team), add the **validators** list of reviewers for a final +2 review followed by a submit to the master branch.

3. Appendix

3.1. Revision marks

Differences since last tag