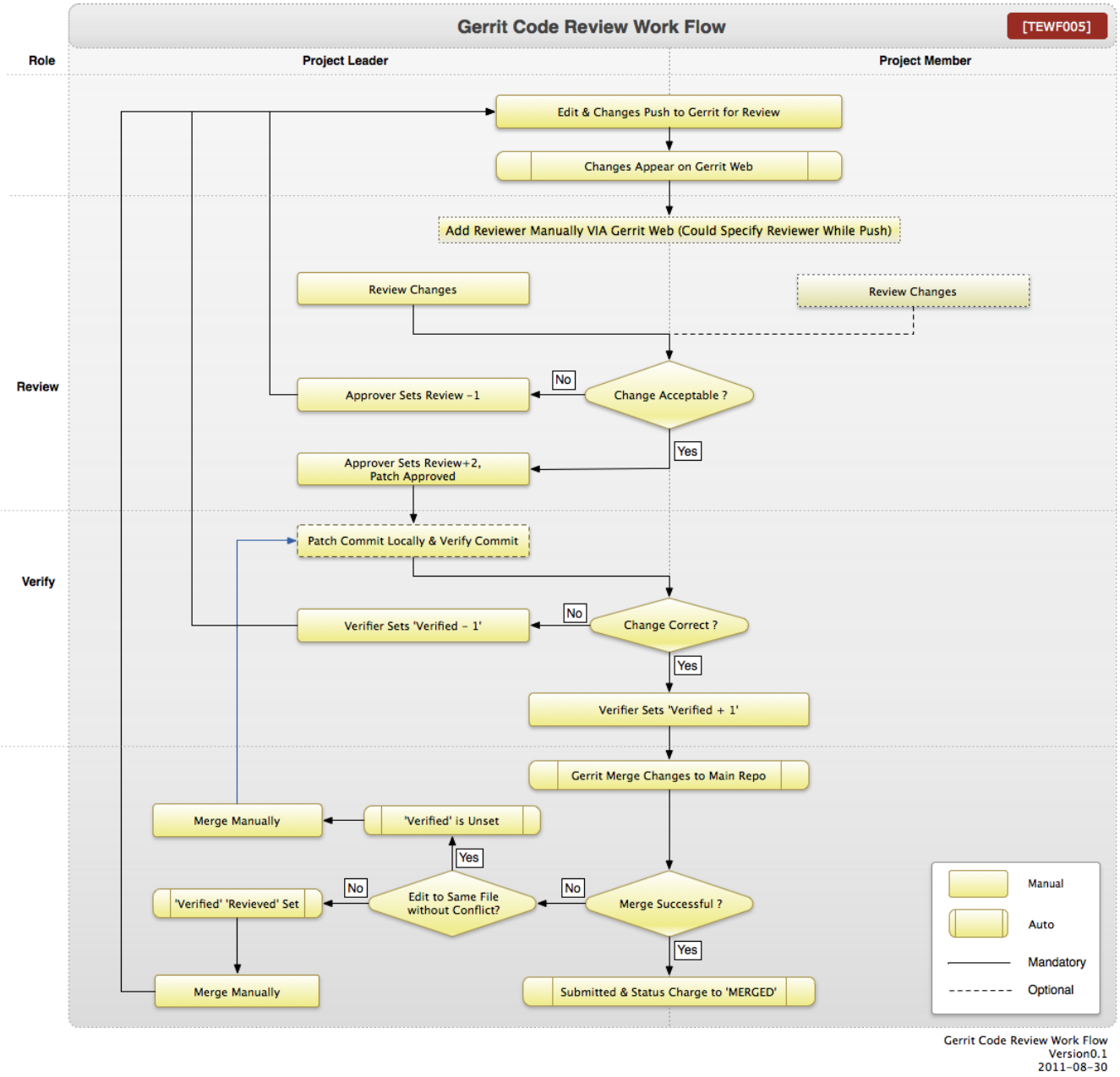
1. Gerrit Code Review Workflow

The graph below illustrates the workflow using gerrit:



Gerrit Code Review Workflow

When developers commit changes into gerrit, gerrit will keep the changes in a virtual branch for review, changes won't be merged into main repository only if they are reviewed and verified.

When a commit is pushed to gerrit, if it's approved and verified, it could be

submit to main repository via gerrit web directly, but if it's not accepted, the original author may need to redo the commit according to the feed backs from the reviewers(See C4' in above graph). Each time a commit is resubmitted, the status will be reset for reviewing again.

2. Status for a review request

1) Code review

+2: Looks good to me, approved

+1: Looks good to me, but someone else must approve

0: No opinion, just adding some comment

-1: Please do not commit

-2: Do not submit(This blocks the submit)

2) Verified

-1: Verify failed

0: Have not tested

+1: Verify succeeded, ready to submit

3) Merged

Indicates the changes are merged in main repo.

4) Abandoned

Indicates the changes are abandoned.

For more details, refer to:

http://hss-tools.englab.nay.redhat.com/gerrit/Documentation/access-control.html

3. Project roles for our team

We'll have 3 groups generally: 'Administrators', 'Registered Users', 'Anonymous Users', 2 roles inside a project: 'Project Leader' in 'Administrator' group and 'Project member' in 'Registered Users' group. Following are the permissions and responsibilities for these groups:

Anonymous Users could view the reviews from gerrit web, but could not make changes on the reviews.

All signed-in users are a member of 'Registered Users' automatically, and could push codes for review, could review other member's code and publish comments(Review -1～+1 permission).

Administrators could approve/deny the review(Review: -2～+2), verify/not verify the reviews(Verify: -1～+1). So administrators in our team should act both as an approver and verifier.

All registered user could submit an 'Approved and Verified' patch, 'Submit' would merge the changes to our main repo, this should be done via gerrit web.

If a merge got conflicts, please follow the instructions described in the flow diagram above for action, you could also refer to following link:

http://source.android.com/source/life-of-a-patch.html

4. Gerrit Configurations

1) Set up an account in gerrit:

Login gerrit with your OpenID account, click 'settings' and update your profile as follows:

username: Your kerberos username.

email: Suggests register redhat email as a “Preferred email”.

Make sure your username/email consistent with your git configurations,

2) Update SSH Public Keys: Add your ssh public key (~/.ssh/id_rsa.pub) to enable ssh, if you don't have a ssh key, you may need to generate one, see github's guide to SSH keys for more details.

3) Enable ssh to gerrit server.

Suggest edit ~/.ssh/config add following lines:

```
Host gerrithost
  User <your username>
  Port 29418
  Hostname hss-tools.englab.nay.redhat.com
  IdentityFile <path/to/private/key>
```

The .ssh/config file should have permissions '600'. Test your ssh connection:

ssh gerrithost

If the connection is made successfully, you could communicate with gerrit project via ssh, if it failed, you may need to double check your configurations.

4). Create a new project in gerrit:

Prerequisite: Have permission to create new project.

Run following:

ssh gerrithost gerrit create-project -n <project-name>

clone the project to local repository:

git clone gerrithost:project-name.git

Get commit msg hook from gerrithost:

cd project-name

scp -p -P 29418 gerrithost:hooks/commit-msg .git/hooks/

5). Init remote branches(master & develop):

Prerequisite: Have permission to push branch.

Login the page, go to project admin page and configure following for your project:

<Forge Identity> +2

<Push Branches> +2

If you already have a local master & develop, run following:

git remote add gerrit ssh://gerrithost/projectname.git

git push gerrit master //create remote branch master

git push gerrit develop //create remote branch develop

or push directly without adding remote:

git push ssh://sshusername@hss-tools.englab.nay.redhat.com:29418/projectname master(or develop)

If you don't have a local branch, create local master/develop and then push to remote.

Note that step 4 and step 5 should be done only once in each project. It's purpose is to create new project and initiate remote branches in gerrit. Later you should avoid pushing your changes directly into those branches. Always push

your changes to HEAD:refs/for/<branch_name> for review(branch_name should be an existing remote branch in gerrit).

5. Create changes for review

To upload your commit in gerrit web, make sure you have copied the 'commit-msg' hook and placed it in your local .git/hooks/ before commit:

scp -p -P 29418 gerrithost:hooks/commit-msg .git/hooks/

1). Edit and commit your local changes

2). Run 'git push origin HEAD:refs/for/<branch-name>' to push local changes as a new review.

For convenient, edit your .git/config add following lines:

[remote "<branch_name>"]

url = gerrithost:<project-name>.git

push = HEAD:refs/for/<branch_name>

For example, following commands configure a new remote 'develop':

git remote add develop ssh://gerrithost/projectname.git

git config –add remote.develop.push HEAD:refs/for/develop

Later by running 'git push develop' will push your changes to gerrit virtual branch 'develop' for review automatically.

It's also possible to specify reviewers while push, details could be found here:
http://hss-tools.englab.nay.redhat.com/gerrit/Documentation/user-upload.html#push_create

3). After you pushed the changes, the result will include an HTTP URL which points to the newly created code review.

Open the URL to view the changes. Any registered user can review your change if they like, but you can also invite people to review the code using the 'Add Reviewer' button if they have an account in gerrit.

Gerrit will use the Change-Id added by the commit message hook to identify the change and will add it as a new patch set on the same code review, resetting the status of the review.

Note that when reviewing a series of related patches, the patches after the first rejected one will need to be reviewed again.

4). If a commit failed to be verified, you may need to re-commit it for the fix, remember to preserve the Change-Id in commit msg, see following example:

C1--------->C2--------->C3
                        \
                        C3'

Redo a commit:

If your commit is not accepted, maybe you'll need to do some fix and commit again.

C1--------->C2--------->C3

Assuming we're going to do another commit C3' for C3:

(1) Edit files and commit your changes.

(2) git rebase -i HEAD~2

(3) In the interactive dialog, change the last commit from 'pick' to 'fixup', by doing this we add another C3' to replace C3, but preserved the Change-Id generated for C3.

(4) Exit and push your changes, if it's successful, another patch will be shown in gerrit web:

We could download the commit using 'checkout', 'pull', 'cherry-pick' or 'patch', commands already listed in gerrit web(see graph above).

If you got errors, refer to:
http://hss-tools.englab.nay.redhat.com/gerrit/Documentation/error-messages.html

How to avoid a merge commit which actually did nothing?

Following are instructions on how to avoid generating the annoying merge commit:

We got our 'develop' branch which track remotes.
We work on out feature branch daily, when we want to push changes:

[1] git checkout develop
[2] git pull origin develop // Pull latest code

[3] git checkout feature-branch // Our feature-branch
[4] git rebase develop //Rebase feature-branch on top of develop
[5] git checkout develop // Switch to develop
[6] git merge feature-branch // Merge feature-branch into develop
[7] push your changes for review.

Your local 'develop' branch should be clean, if it's not clean, you should use 'git pull --rebase' in [2] to avoid merge commit.

Following the steps above, we merged feature-branch into develop without generating
additional merge commit. This makes our commit history looks nice.

An example of rebasing

Our scenario:
I'll take an 'openshift' project as an example, suppose I have a clean repository now, I'll give four commits by run following:

```
$ for i in 1 2 3 4
do
   echo “Append one line in README(commit $i)” >> README
   git add README
   git commit -am “Commit $i”
done
```

See what I did locally:

```
[yuwang@yuwang openshift(develop)]$ git diff origin/develop..develop
diff --git a/README b/README
index 75f37d5..64e59db 100644
--- a/README
+++ b/README
@@ -50,3 +50,7 @@ Notes about setup.py

 Adding deps to the install_requires will have the openshift server actually
 install those deps at git push time.
+Append one line in README(commit 1)
+Append one line in README(commit 2)
+Append one line in README(commit 3)
+Append one line in README(commit 4)
```

git log --pretty=oneline --abbrev-commit

The output:

```
cd3d126 Commit 4
11290bd Commit 3
a9ed980 Commit 2
bbe4fc9 Commit 1
```

By running

git push origin bbe4fc9:refs/for/develop

Where “bbe4fc9” is commit 1's hash, I've managed to push “Commit 1” into gerrit for review.

But after peer review, I was told I did some thing silly which may cause regression in “Commit 1”, I need to do another commit to fix the hidden problems, so I create a branch which use “Commit 1” as a start point and made another commit(see 'git branch –help' for more details):

git checkout -b fix-branch bbe4fc9

Open README and did some fix:

```
[yuwang@yuwang openshift(fix-branch)]$ git diff
diff --git a/README b/README
index 22ab0ce..94c27ca 100644
--- a/README
+++ b/README
@@ -50,4 +50,5 @@ Notes about setup.py

 Adding deps to the install_requires will have the openshift server actually
 install those deps at git push time.
-Append one line in README(commit 1)
+Append one line in README(commit 1) Do something here to fix the hidden problems
+in previous commit.
```

Now let's push our commit into gerrit(I'm currently on 'fix-branch'):

git rebase -i HEAD~2

In the pop-up interactive dialog, you would see this:

```
pick bbe4fc9 Commit 1
pick 960a9e8 Fix commit 1's problems

# Rebase c04782d..960a9e8 onto c04782d
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x <cmd>, exec <cmd> = Run a shell command <cmd>, and stop if it fails
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Let's replace “pick” in the 2nd line to be 'fixup', see following:

```
pick bbe4fc9 Commit 1
f 960a9e8 Fix commit 1's problems

# Rebase c04782d..960a9e8 onto c04782d
#
# Commands:
#  p, pick = use commit
#  r, reword = use commit, but edit the commit message
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#  f, fixup = like "squash", but discard this commit's log message
#  x <cmd>, exec <cmd> = Run a shell command <cmd>, and stop if it fails
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Save and quit. Let's see what happened:

git log

```
commit ed7c29dc1817a88a11d2f9b90add6758f09ec88f
Author: Yuguang Wang <yuwang@redhat.com>
Date:   Wed Oct 26 15:28:43 2011 +0800

    Commit 1

    Change-Id: I464d7727f3dd46f595f98915977a0c86e614b00e
```

Let's see what changes are included in this commit:

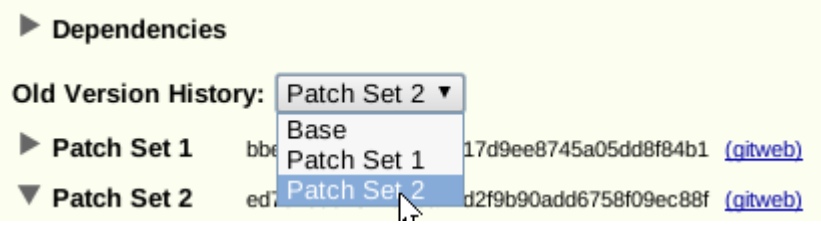
git show HEAD

```
[yuwang@yuwang openshift(fix-branch)]$ git show HEAD
commit ed7c29dc1817a88a11d2f9b90add6758f09ec88f
Author: Yuguang Wang <yuwang@redhat.com>
Date:   Wed Oct 26 15:28:43 2011 +0800

    Commit 1

    Change-Id: I464d7727f3dd46f595f98915977a0c86e614b00e

diff --git a/README b/README
index 75f37d5..94c27ca 100644
--- a/README
+++ b/README
@@ -50,3 +50,5 @@ Notes about setup.py

 Adding deps to the install_requires will have the openshift server actually
 install those deps at git push time.
+Append one line in README(commit 1) Do something here to fix the hidden problems
+in previous commit.
```

After rebasing, we combined previous commit and our new commit into one commit, and preserved the Change-Id line.
Now let's push it into gerrit by running:
  git push origin ed7c29d:refs/for/develop

That's all, now we have another patchset available:

http://hss-tools.englab.nay.redhat.com/gerrit/#change,106



There's many ways of re-do a commit and preserve Change-Id line in git. You could use cherry-pick, or manually maintain the Change-Id etc.
Feel free to contact me if you got any problems.