



**CMMIDEV / 3**<sup>SM</sup>  
Exp. 2018-02-28 / Appraisal #23707

# Git-Gerrit Practice Session

Prepared by: Sajidul Huq Rubon  
Date: 25 Jul, 2018



- ❖ It mainly focuses on the usage of git and gerrit and discussed about the best practices.
- ❖ This document is mainly designed for practical sessions and practices.
- ❖ Please understand that our knowledge about git is still very low. During this practice session we will try find out better solutions that we are facing everyday.
- ❖ This is not any training session, this is basically a discussion and practice session to fix git-gerrit usage problems.
- ❖ This document also doesn't cover commit message rules or branching rules.
- ❖ This document is prepared based on Linux system, on other platforms it may vary slightly.

- Git Installation
- Understanding what is git
- Download remote repository – git clone
- Create, delete and checkout local branch
- Git Commit and Push
- Understanding Gerrit
- Working style – best practice
- Upload changes for review – git push
- Before upload merge with latest code
- Fixing merge conflict
- Update your changes – commit amend
- Update your changes by additional patch-set
- Merge failed on review server
- Commit that will dependent on another pending commit
- Tag your releases

# Git Installation (1/2)

- Download Git client from <https://git-scm.com/downloads>
- Install at your workstation, follow the default options.
- After the installation you need to create SSH key and upload the public key to Gerrit/Review server
- Now in order to create .gitconfig for client, type the following 2 commands and press
  - git config --global user.name "User's Full Name"
  - git config --global user.email user.name@bjitgroup.com
- For example, for user sajidul.huq following commands will be used.
  - git config --global user.name "Sajidul Huq Rubon"
  - git config --global user.email sajidul.huq@bjitgroup.com

## ➤ Creating SSH key

- Now from the GIT Bash command prompt or Windows default command prompt, type the following command.

**ssh-keygen -t rsa**

- It will say “Enter file in which to save...” , don't type anything, press Enter
  - It will again ask for Passphrase, don't type anything, press Enter
  - It will again ask for confirmation on Passphrase, don't type anything, press Enter
  - Note: Please do not input passphrase / password during key file generation.
  - After completing aforementioned steps public key will be saved in a file under .ssh folder, for example “C:\Users\Rubon\.ssh\id\_rsa.pub” file
- ## ➤ Adding SSH Key to review server
- Go to review server from your browser, <https://review2.bjitgroup.com:8443/>
  - Sign in by using your LDAP credentials
  - Click on 'Settings',
  - Now from the left pane, click on “SSH Public Keys”,
  - Now from the bottom of the text area, click “Open Key ...” button,
  - Select 'id\_rsa.pub' file which you created earlier under the .ssh folder located at user
  - directory and press 'open' button.
  - Now press 'Add' button, SSH key will be added.

# Download remote repository – git clone

- Git Clone means download remote repository in your local drive.
- Clone command

```
git clone ssh://<user.name>@review2.bjitgroup.com:29418/<project_name>
```
- Example:

```
git clone ssh://sajidul.huq@review2.bjitgroup.com:29418/test
git clone ssh://sajidul.huq@review2.bjitgroup.com:29418/p0962_newns
```
- The above command will download the repository on your current directory.
- If you want to download in a different directory, just mention the directory name or path in the command

```
git clone ssh://<user.name>@review2.bjitgroup.com:29418/project_name <local_path>
```
- Example:

```
git clone ssh://sajidul.huq@review2.bjitgroup.com:29418/test NewFolder
git clone ssh://sajidul.huq@review2.bjitgroup.com:29418/test
/home/rubon/workspace/AndroidProject/NewNS/Source
```

# Local branch – Create, Delete, Checkout (1/2)

- Check the existing branch list.

```
git branch          // This will show all the local branches.  
git branch -r       // This will show all the remote branches.  
git branch -a       // This will show all the local and remote
```

branches.

- Note the asterisk (\*) on master branch. This means you are currently in master branch.
- Create a local branch
  - Example:
- Now you can check the branch list again.
- Please note that you are still on your previous branch.
- To switch to your new branch you need to checkout to that branch.

```
git checkout my_local_branch
```

```
rubon@Rubon: /media/Projects/Proje  
rubon@Rubon:/media/Projects/Projects/tes  
t/newfolder$ git branch -a  
* master  
  remotes/origin/HEAD -> origin/master  
  remotes/origin/foo2  
  remotes/origin/foo3  
  remotes/origin/hello_branch  
  remotes/origin/master  
  remotes/origin/testbranch  
rubon@Rubon:/media/Projects/Projects/tes  
t/newfolder$
```

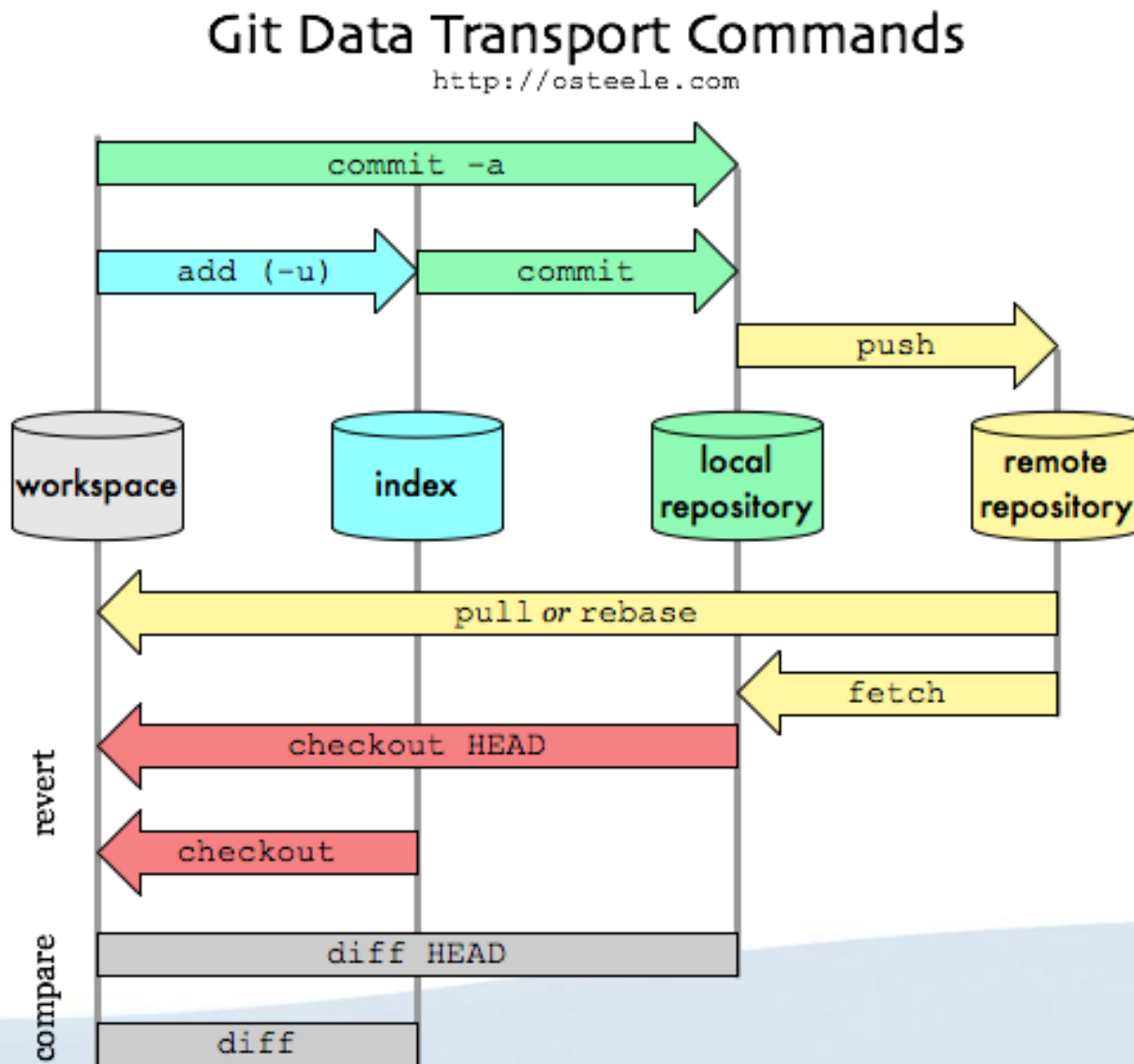
```
rubon@Rubon: /media/Projects/Proje  
rubon@Rubon:/media/Projects/Projects/tes  
t/newfolder$ git branch my_local_branch  
rubon@Rubon:/media/Projects/Projects/tes  
t/newfolder$ git branch  
* master  
  my_local_branch  
rubon@Rubon:/media/Projects/Projects/tes  
t/newfolder$
```

# Local branch – Create, Delete, Checkout (2/2)

- To create new branch and checkout to that branch in a single command  
`git checkout -b my_local_branch`
- Delete the created local branch  
`git branch -d my_local_branch`
- Note that you cannot delete the branch that you are currently in.
- You may not able to delete a branch if it has any changes that is not committed yet. If you still want to delete that branch, use the following command.  
`git branch -D my_local_branch`



# Git Basics (Workspace, Index, local repo, remote repo)



- Stores the current contents of the index in a new commit along with a log message from the user describing the changes.

`git commit <options>`

- Before commit you need to add your changed files into stage for commit. Only staged files will be committed.

`git add <filenames>`

`git add Utility.java`

// adds Utility.java into stage for commit

`git add LoginController.java LoginView.java`

// adds both classes

`git add -a`

// adds all tracked files

- Please note that this will not add your untracked or new files

`git add -A`

// adds all tracked and untracked files

`git reset <filenames>`

// remove file from stage

- You can use the following command anytime to checkout your repository status.

`git status`

- When you are ready for commit, commit using the following command.

`git commit -m "<Your commit message here>"`

- To add all of your tracked files into stage and commit in a single command

`git commit -a -m "<Your commit message here>"`

# Git Commit (2/2)

- You may need to update your last commit. You may don't want to make new commit, instead you need to update your last commit. Following command would work.

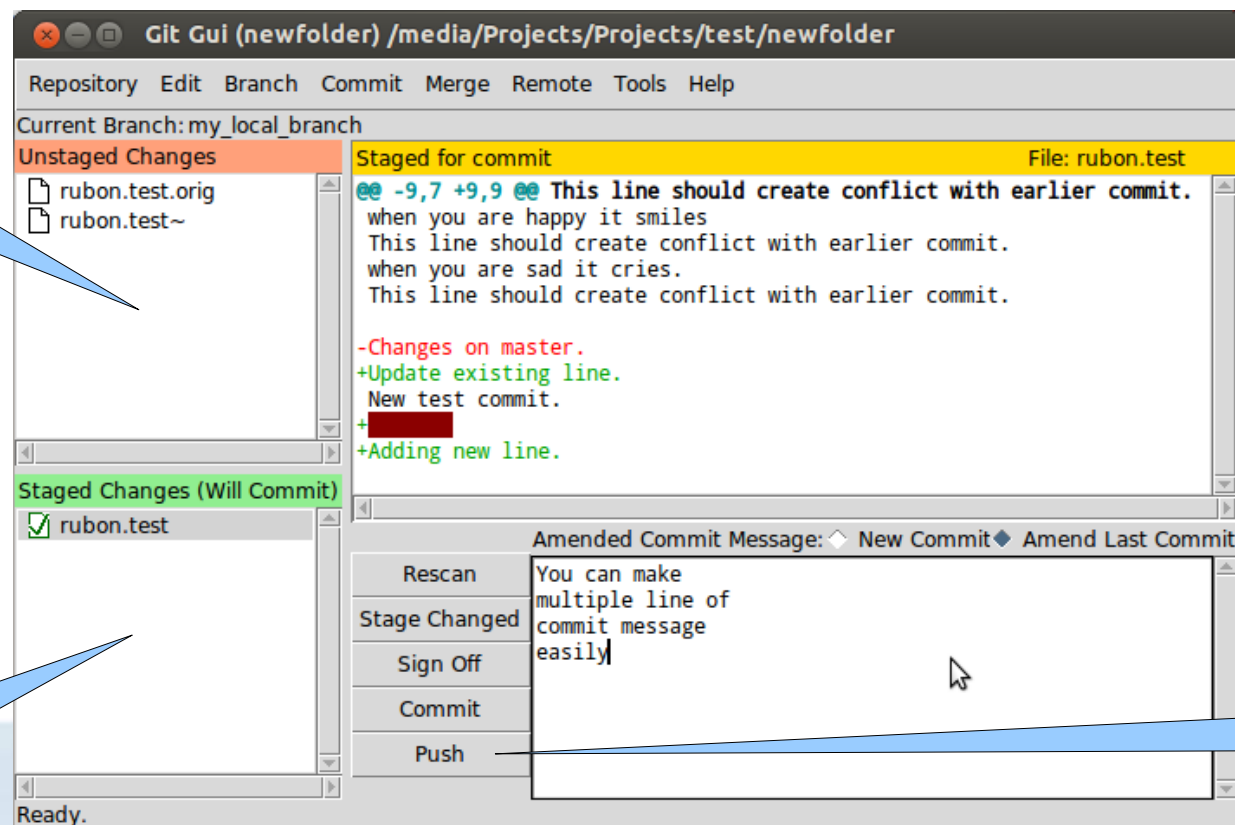
`git commit -a --amend -m "<Your commit message here>"`

- I personally prefer to use gui tool for this purpose, **Git GUI**.
- Using git gui you can easily add/remove files from stage, can see your changes, red marks, write multiple line of commit message and can commit. **But you should not push from git gui.**

`git gui`

Unstaged files  
Will not commit

Staged files  
Will commit



Don't push  
from here

- Please understand that commit is stored in your local system. Your changes still in your PC only.
- Now you need to upload your changes for review to the review server.  
`git push origin HEAD:refs/for/<remote_branch_name>`  
`git push origin HEAD:refs/for/master`  
`git push origin HEAD:refs/for/phase2`
- After push, go to the review server, <https://review2.bjitgroup.com:8443>, check your changes there.
- If you think your commit is okay and the reviewer can review now, please add reviewer from the review server.
- Suppose Hodayun Kabir is the TL/PM of your project and you want him to review your code. Then add him as reviewer if you feel that your commit is ready for review.

Reviewer	Verified	Code Review
<a href="#">Sajidul Hug</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

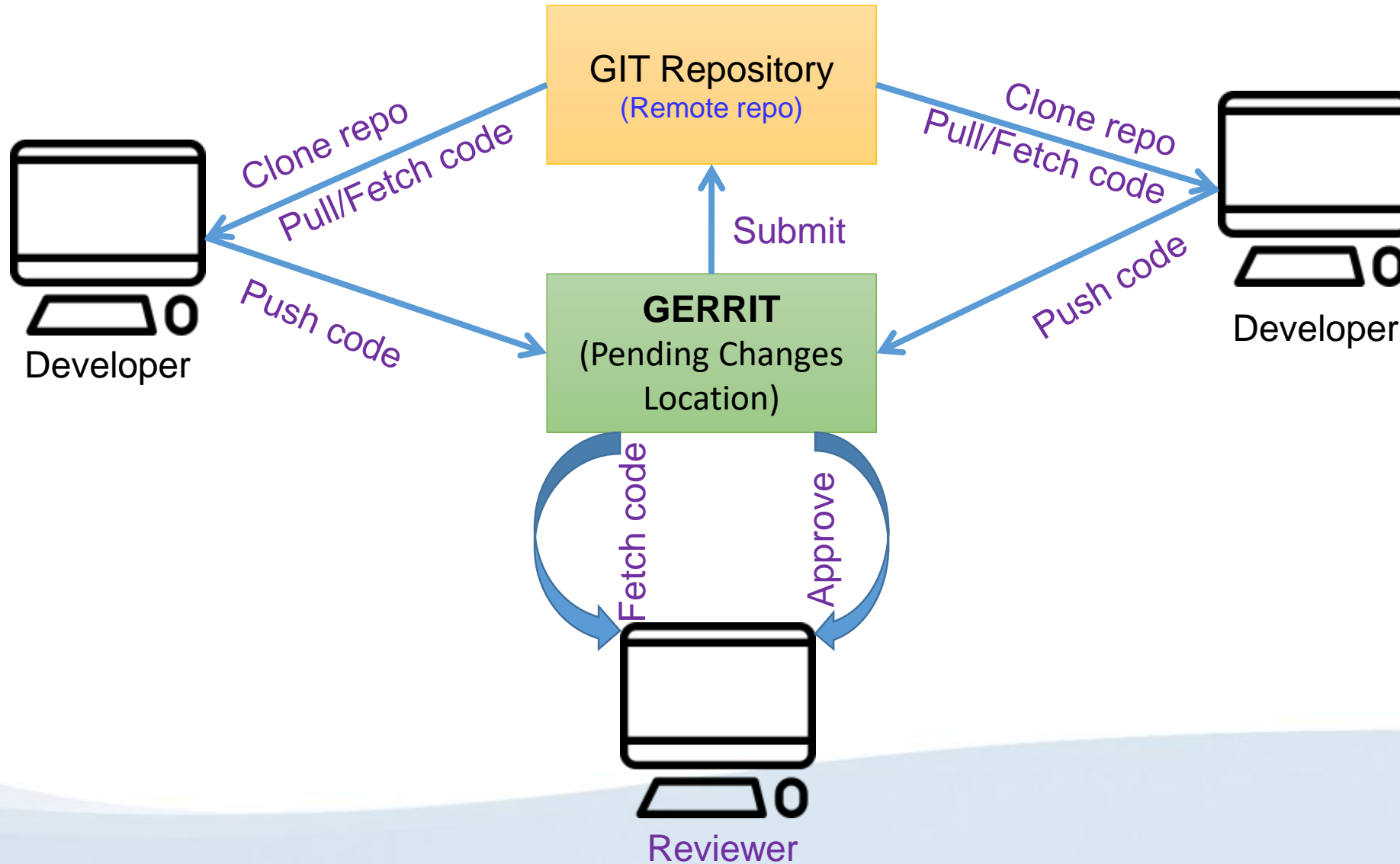
- Need Verified +1 (Verified)
- Need Code Review +2 (Looks good to me, approved)

Hodayun Kabir <hodayun.kabir@bjitgroup.com>

Reviewer	Verified	Code Review
<a href="#">Sajidul Hug</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<a href="#">Hodayun Kabir</a>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

- Need Verified +1 (Verified)
- Need Code Review +2 (Looks good to me, approved)

# Understanding Gerrit



# Working guideline – Best Practices (1/2)

- Don't work on master branch, or your main branch. Work only in your local branch.
- Create your local branch for a specific task.
- After finishing your task, commit it and push it to master branch(if you want to push into master).
- For every separate tasks or tickets create separate local branches, work on that branch, commit and push to master.
- If there is any review feedback, switch to that branch, fix and commit additional patch-set.
- Lets see the steps of a total work flow:
  - `git clone or git pull` // to take latest update
  - `git checkout -b iss#303` // create and checkout to local branch
  - Finish your task
  - `git commit` // only commit, don't push
- Always keep your master or main development branch clean. This will help you to take latest updated source by git pull.
- Lets think about another scenario. Suppose you are working on a feature in master branch. During your development you receive another high priority bug fixing task. What will you do?
- The simple solution would be if you work on your local branch. Commit your current incomplete task in your local branch. Create/Checkout to the high priority bug task branch, finish it, push it. Now back to your earlier feature task local branch, finish your task, amend a commit and push.

# Working guideline – Best Practices (2/2)

- After you commit your changes you need to follow the bellow steps:
- Assumption: You are now on your local branch and committed your changes.
  - `git checkout master` // checkout to master branch
  - `git pull` // get latest source in master
  - `git checkout iss#303` // checkout to your working local branch again
  - `git merge master` // merge your changes with latest code
  - `git mergetool` // if auto merge failed then use this command
  - `git commit` // it may push 2 commits, one empty commit for merge
  - `git push`
- Now from review server, first merge the empty or merge commit. Then merge the original/desired commit.
- **Isn't this empty commit seems to be a problem, specially for the reviewer!! Please keep patience, I will show you a workaround of this problem soon.**

- There are some gui mergetool, but I would recommend KDiff3, a 3 way mergetool.
- Download and install KDiff3 from <http://kdiff3.sourceforge.net/>
- After installing KDiff3, run the bellow commands to configure your git.

```
git config --global merge.tool kdiff3
```

```
git config --global mergetool.kdiff3.cmd "kdiff3" // for Linux users
```

```
git config --global mergetool.kdiff3.cmd "'C:\\Program Files (x86)\\KDiff3\\kdiff3" $BASE $LOCAL  
$REMOTE -o $MERGED'
```

- After that your .gitconfig file should look like this:

```
[user]
```

```
name = Sajidul Huq Rubon
```

```
email = sajidul.huq@bjitgroup.com
```

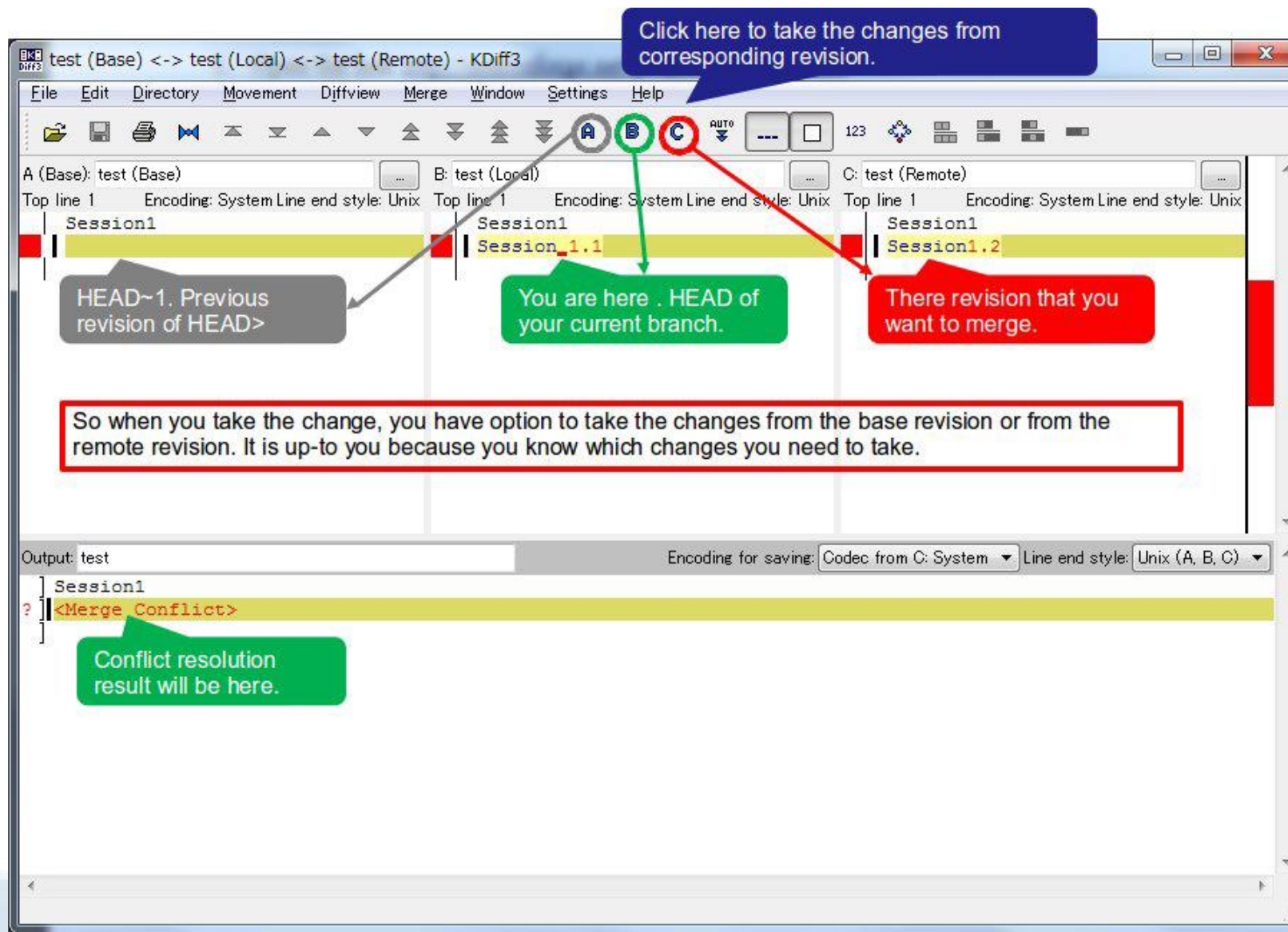
```
[merge]
```

```
tool = kdiff3
```

```
[mergetool "kdiff3"]
```

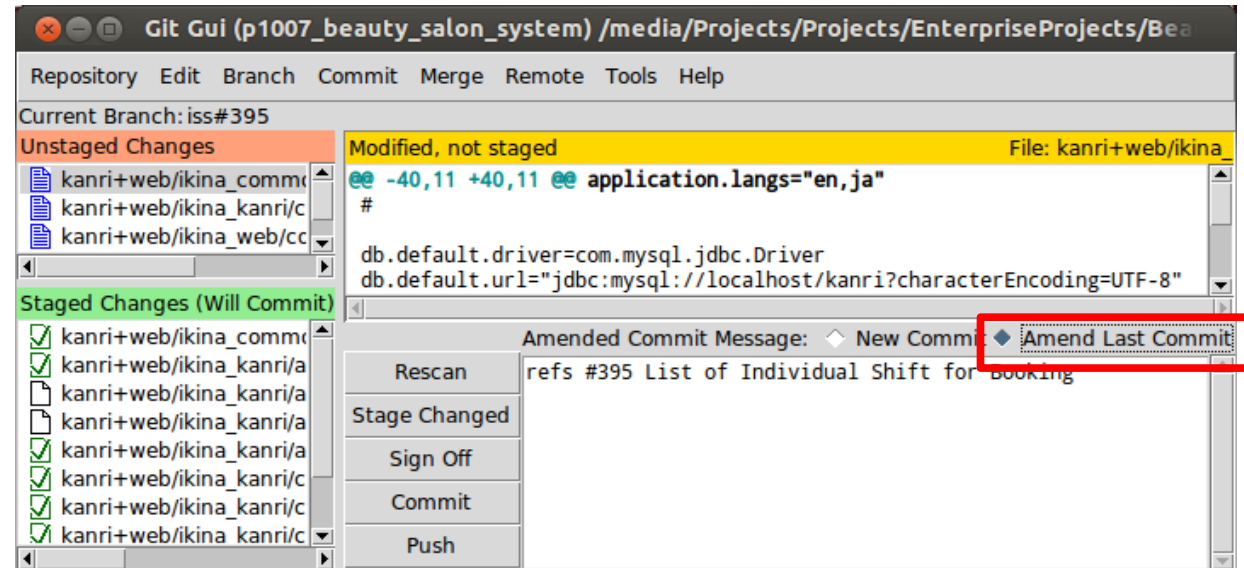
```
cmd = kdiff3 $BASE $LOCAL $REMOTE -o $MERGED
```






# Bjit Update your commit – commit amend (1/2)

- You may need to update your commit for various reasons:
  - You may find a problem/bug by yourself. So you need to update your commit.
  - Reviewer might give review feedback that you need to fix. So you need to update your commit.
  - For any reason you checkout to different branch without finishing your current task. So you committed partial task. Now you come back to that work, finish your task. So you need to update your commit.
- You can update your last commit very easily. Here is the command.  
`git commit -a --amend -m "Write your comment here"`
- But it's much more simple using git gui:



# Bjit Update your commit – commit amend (2/2)

- If you have different local branches for each task/commit, you can easily update your commit by this way.
  - Checkout to that branch, update your commit and amend last commit.
- If you do not have local branch, but you may need to update a commit.
  - Create a local branch from master and checkout to that branch.
  - Now cherry-pick that particular commit from the review server. Don't forget to select cherry-pick from Download options.
  - Update your commit and amend last commit.

Author	<a href="#">Sajidul Huq Rubon</a> <sajidul.huq@bjitgroup.com> Feb 13, 2014 2:32 PM				
Committer	<a href="#">Sajidul Huq Rubon</a> <sajidul.huq@bjitgroup.com> Feb 13, 2014 2:32 PM				
Download	checkout	pull	cherry-pick	patch	SSH HTTP
git fetch ssh://sajidul.huq@review.bjitgroup.com:29418/p1007_beauty_salon_system refs/changes/90/14390/9 && git cherry-pick FETCH_HEAD 					

ReviewAbandon ChangeDiff All Side-by-SideDiff All Unified

# Upload your updates – patch-set

- Consider that you have made a commit and uploaded to review server for review. Now you need to update your commit.
- You have updated and amend your last commit. Now you need to push your commit to review server.
- If you push like new commit push(git push origin HEAD:refs/for/master), that you create a new commit. That means you need to abandon your earlier commit from review server.
- So reviewer need to check the abandoned commit list to check his last time feedback or comments.
- You can easily push for review with the existing commit so that you/reviewer can see all your commits for a single task together.
- For this what you need to do:
  - Go to your commit on review server that you want to update.
  - You should see an URL like this: `https://review2.bjitgroup.com:8443/#change,<14390>`
    - Here **14390** is the **change-id**.
  - Now use the below command to upload additional patch-set:  
`git push origin HEAD:refs/changes/<change-id>`  
Example: `git push origin HEAD:refs/changes/14390`
- Now got back to review server, you should find additional patch “Patch Set 2”.

# How to fix merge failed on review server (1/2)

## Gerrit Code Review

11:38 AM

Your change could not be merged due to a path conflict.

Please merge (or rebase) the change locally and upload the resolution for review.

- I guess we all are well known of this error message, yes, this is the path conflict error on Gerrit code review.
- This mainly occurs if you have changes in X file but master(or your desired branch) has update on the same file.
- Now lets see how can we fix this problem.
  - We already know that we always should work on local issue branch, not in master branch. Master branch should kept untouched. So I assume that your master branch doesn't have any change.
  - First switch back to your master branch if you are in different branch. Before switch back make sure that all changes in your current branch is committed.  
`git checkout master`
  - Make sure that your master branch doesn't have any local changes,  
`git status`
  - Take latest code update on master.  
`git pull`
  - Create a new local branch from master and checkout to that branch.  
`git checkout -b iss#102-fixconflict`

# How to fix merge failed on review server (2/2)

- Now cherry-pick the commit that raised path conflict. You can copy paste the cherry-pick command from review server.

```
git fetch ssh://user.name@review.bjitgroup.com:29418/test refs/changes/82/12282/1 && git  
cherry-pick FETCH_HEAD
```

- This may auto merge your code with latest source or merge conflict may arise. If merge conflict arise, fix it locally using mergeconflict(KDiff3).

```
git mergetool
```

- Now commit your code and push to review server again. For push, please submit additional patch-set.
- Now this should be merged without any conflict.



# Workaround for merge with empty commit

- In Working Style-Best practices section, we saw how to merge with latest code before we upload the commit. But it has one problem, it pushes two commits, where one commit is empty.
- We can fix this problem similarly we fix the merge conflict error in last slide.
- Here are the steps in short.
  - Finish your task in local branch, commit and push.
  - Checkout to master, take latest source by git pull.
  - Create a new local branch from master and checkout to that branch.  
`git checkout -b iss#102-fixconflict`
  - Now cherry-pick the commit that you just pushed. You can copy paste the cherry-pick command from review server.  
`git fetch ssh://user.name@review.bjitgroup.com:29418/test refs/changes/82/12282/1 && git cherry-pick FETCH_HEAD`
- This may auto merge your code with latest source or merge conflict may arise. If merge conflict arise, fix it locally using mergeconflict(KDiff3).  
`git mergetool`
- Now commit your code and push to review server again. For push, please submit additional patch-set.
- This will show only a single commit, there will be no empty commit. And this code is merged with latest. :)

# Make a commit that will depend on a pending commit (1/2)

- Your current task may depend on another task. There may two scenarios:
  - It depends on one of your own task, may be your last commit
  - It depends on a task that another team member committed.
- For the **first case** its simple and straight forward.
- Just create the new ticket/issue local branch from the dependent local branch.
- Example, you finished your task in iss#101 branch. Now you want to work on next issue 102 that has dependency with last task 101. So first checkout to iss#101 branch(`git checkout iss#101`) and then create new branch iss#102(`git checkout -b iss#102`).
- For **second case**, let me describe the problem scenario first.
- After finishing your task you need to commit. Now you find that there is another pending commit on review server waiting for review. And your commit should be dependent on that commit because:
  - you both worked in the same file or
  - have dependency on that changes or
  - dependent commit created a new class and you want to modify on that new class
  - for other reasons.
- That means your commit cannot (or should not) be merged until the pending commit is merged. But also don't want to wait for the pending commit to be merged. Infact you should not wait.
- There is solution for this problem. You can make your commit that will be dependent on the pending commit. So your commit cannot be merged until the pending commit is merged.



# Make a commit that will depend on a pending commit (2/2)

- 1. First create a local branch for your task.  
`git checkout -b iss#102-temp`
- Now finish your task/issue in this local branch("iss#102-temp") and commit. **DON'T push.**
- 2. Now go to the pending commit on review server. You should see a checkout command for that particular pending commit, copy this command. Now paste this command in your working branch with creating a new branch from this checkout.  
`git fetch ssh://sajidul.huq@review.bjitgroup.com:29418/test refs/changes/09/12209/1 && git checkout FETCH_HEAD -b iss#102`
- That means you have checked out the pending commit in "iss#102" local branch. That means your "iss#102-temp" branch is still the same. This also means that you are now on "iss#102" branch.
- 3. Now you need to cherry-pick your changes that you made in iss#102-temp branch.  
`git cherry-pick SHA1` // SHA1 id of your commit in "iss#102-temp" branch OR  
`git cherry-pick iss#102-temp` // if iss#102-temp has only a single commit
- 4. There may merge conflict after cherry-pick. If have merge conflict, please use mergetool to fix the conflict.  
`git mergetool`
- 5. After fixing, commit amend the merged result. And PUSH to review server. See the dependency of your commit, it should be dependent on the pending commit.
- NOTE: The patch set of the pending commit may not be approved, the owner of that commit may need to submit additional patch set. If he submit additional patch set, you will also need to submit additional patch set. To do this, follow the same above steps 2-5.

# Git Configurations – Terminal Colours (Optional)

- Git can color its output to your terminal, which can help you visually parse the output quickly and easily. Git automatically colors most of its output if you ask it to. You can get very specific about what you want colored and how; but to turn on all the default terminal coloring, set color.ui to true:

```
git config --global color.ui true
```

- Tag is created to tag a release.
- To see existing tag list use the below command  
`git tag -l`
- To create a new tag use the below command  
`git tag -a <version_name> <sha1_id> -m "Write your message"`
- After that you need to push the tag  
`git push origin <version_name>`
- As for example:  
`git tag -a "beta_release" -m "Beta release is tagged"`  
`git push origin "beta_release"`

# Git Compare tags or branches

- Tag is created to tag a release.
- To see existing tag list use the below command  
`git tag -l`
- To create a new tag use the below command  
`git tag -a <version_name> <sha1_id> -m "Write your message"`
- After that you need to push the tag  
`git push origin <version_name>`
- As for example:  
`git tag -a "beta_release" -m "Beta release is tagged"`  
`git push origin "beta_release"`

- We have a Forum in Redmine for this purpose. If you have any question or face any problem, please post your question in Redmine.
- <http://redmine.bjitgroup.com/redmine/projects/p0001-6-skill-development/boards>
- We all know about knowledge in git. So please don't hesitate to post your question in the forum. Don't worry that your question might be silly.
- Please understand that your post in the forum will help another engineer to fix his problem. So instead of asking personally, please use this forum.
- Don't forget to reply whether the provided solution worked for you or not.
- I strongly recommend to use this forum.

# Thank You!