



Gerrit

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About the Tutorial

Gerrit is a web-based code review tool, which is integrated with Git and built on top of Git version control system (helps developers to work together and maintain the history of their work). It allows merging changes to Git repository when you are done with the code reviews.

Audience

This tutorial will help beginners learn the basic functionality of Gerrit tool. After completing this tutorial, you will find yourself at a moderate level of expertise in using Gerrit tool from where you can take yourself to the next levels.

Prerequisites

We assume that you are going to use Gerrit to handle all levels of Java and Non-Java projects. Hence, it will be good if you have some amount of exposure to software development life cycle and a working knowledge of developing web-based and non-web-based applications.

Copyright & Disclaimer

© Copyright 2017 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright & Disclaimer	i
Table of Contents	ii
1. Gerrit – Overview.....	1
SETTING UP GERRIT	3
2. Gerrit – Installation.....	4
Installation of Git Client.....	4
3. Gerrit – Configure Git.....	5
4. Gerrit – Set Username & Email.....	6
SET UP SSH KEYS IN GERRIT.....	7
5. Gerrit – Generate New SSH Key	8
6. Gerrit – Add Your SSH Key.....	9
7. Gerrit – Add SSH Key to Your Gerrit Account	11
8. Gerrit – Add SSH Key to Use with Git	13
9. Gerrit – Download Examples Using Git.....	14
PREPARE TO WORK WITH GERRIT	15
10. Gerrit – Installing Git-Review	16
11. Gerrit – Configuring Git-Review	18
12. Gerrit – Setting Up Git-Review	19
HOW TO SUBMIT A PATCH.....	20
13. Gerrit – Update Master.....	21
14. Gerrit – Create a Branch.....	22
15. Gerrit – Make & Commit Your Change	23
16. Gerrit – Prepare Push Change Set to Gerrit	26
17. Gerrit – Push Your Change Set to Gerrit	27

18. Gerrit – View the Change / Next Steps.....	28
19. Gerrit – Editing via the Web-Interface.....	30
HOW CODE IS REVIEWED IN GERRIT	35
20. Gerrit – Review before Merge.....	36
21. Gerrit – Project Owners	37
22. Gerrit – How to Comment, Review, and Merge.....	39

1. Gerrit – Overview

Gerrit is a web-based code review tool, which is integrated with Git and built on top of Git version control system (helps developers to work together and maintain the history of their work). It allows to merge changes to Git repository, when you are done with the code reviews.

Gerrit was developed by Shawn Pearce at Google which is written in Java, Servlet, GWT (Google Web Toolkit). The stable release of Gerrit is 2.12.2 and published on March 11, 2016 licensed under *Apache License v2*.

Why Use Gerrit?

Following are certain reasons, why you should use Gerrit.

- You can easily find the error in the source code using Gerrit.
- You can work with Gerrit, if you have regular Git client; no need to install any Gerrit client.
- Gerrit can be used as an intermediate between developers and git repositories.

Features of Gerrit

- Gerrit is a free and an open source Git version control system.
- The user interface of Gerrit is formed on Google Web Toolkit.
- It is a lightweight framework for reviewing every commit.
- Gerrit acts as a repository, which allows pushing the code and creates the review for your commit.

Advantages of Gerrit

- Gerrit provides access control for Git repositories and web frontend for code review.
- You can push the code without using additional command line tools.
- Gerrit can allow or decline the permission on the repository level and down to the branch level.
- Gerrit is supported by Eclipse.

Disadvantages of Gerrit

- Reviewing, verifying, and resubmitting the code commits, slows down the time to market.
- Gerrit can work only with Git.
- Gerrit is slow and it's not possible to change the sort order in which changes are listed.
- You need administrator rights to add repository on Gerrit.

Setting Up Gerrit

2. Gerrit – Installation

Before you can use Gerrit, you have to install Git and perform some basic configuration changes. Following are the steps to install Git client on different platforms.

Installation of Git Client

Linux

You can install the Git on Linux by using the software package management tool. For instance, if you are using Fedora, you can use as:

```
sudo yum install git
```

If you are using Debian-based distribution such as Ubuntu, then use the following command:

```
sudo apt-get install git
```

Windows

You can install Git on Windows by downloading it from the Git website. Just go to msysgit.github.io link and click on the download button.

Mac

Git can be installed on Mac using the following command:

```
brew install git
```

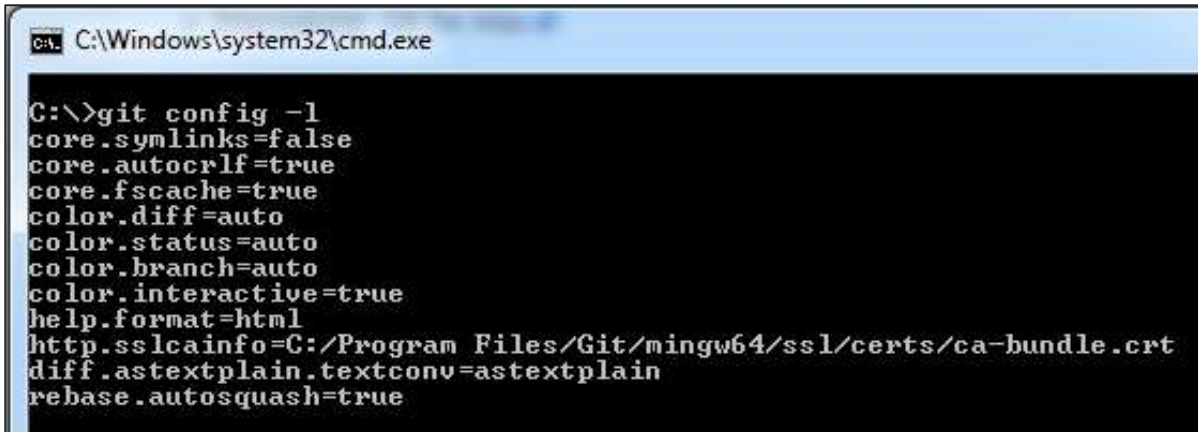
Another way of installing Git is, by downloading it from Git website. Just go to [Git install on Mac](#) link, which will install Git for Mac platform.

3. Gerrit – Configure Git

Once you have installed Git, you need to customize the configuration variables to add your personal information. You can get and set the configuration variables by using Git tool called *git config* along with the *-l* option (this option provides the current configuration).

```
git config -l
```

When you run the above command, you will get the configuration variables as shown in the following image.



```
C:\Windows\system32\cmd.exe

C:\>git config -l
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
rebase.autosquash=true
```

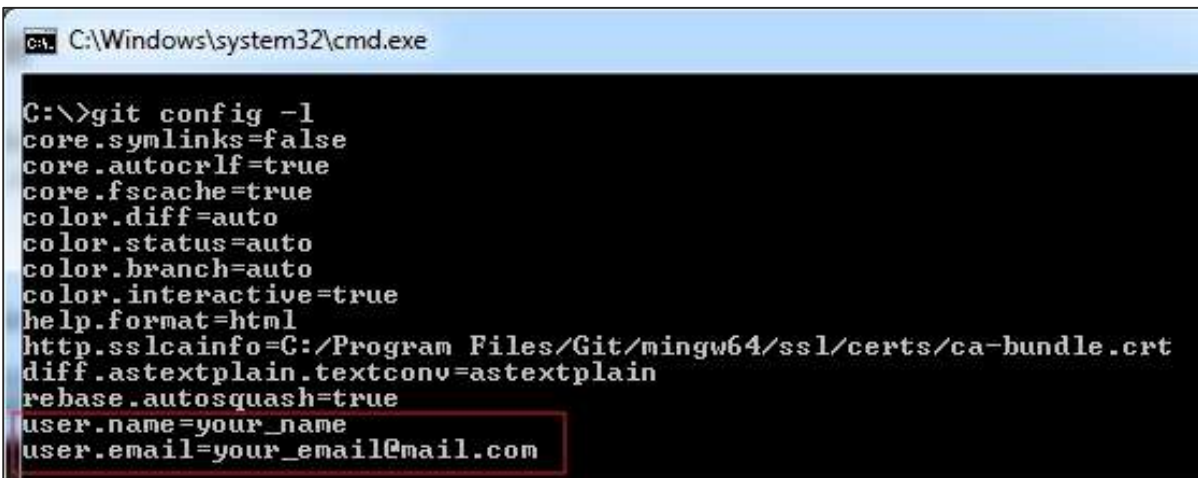
You can change the customized information any time by using the commands again. In the next chapter, you will learn how to configure the user name and user Email by using *git config* command.

4. Gerrit – Set Username & Email

You can track each commit by setting *name* and *email* variables. The *name* variable specifies the name, while the *email* variable identifies the email address associated with Git commits. You can set these using the following commands:

```
git config --global user.email "your_email@mail.com"
git config --global user.name "your_name"
```

When you run the above commands, you will get the user name and email address as shown in the following image.



```
C:\Windows\system32\cmd.exe

C:\>git config -l
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
rebase.autosquash=true
user.name=your_name
user.email=your_email@mail.com
```

Set Up SSH Keys in Gerrit

5. Gerrit – Generate New SSH Key

SSH stands for *Secure Shell* or sometimes *Secure Socket Shell* protocol used for accessing network services securely from a remote computer. You can set the SSH keys to provide a reliable connection between the computer and Gerrit.

You can check the existing SSH key on your local computer using the following command in Git Bash:

```
$ ls ~/.ssh
```

After clicking the enter button, you will see the existing SSH key as shown in the following image:



If you don't find any existing SSH key, then you need to create a new SSH key.

Generating New SSH Key

You can generate a new SSH key for authentication using the following command in Git Bash:

```
$ ssh-keygen -t rsa -C "your_email@mail.com"
```

If you already have a SSH key, then don't generate a new key, as they will be overwritten. You can use `ssh-keygen` command, only if you have installed Git with Git Bash.

When you run the above command, it will create 2 files in the `~/.ssh` directory.

- **~/.ssh/id_rsa**: It is private key or identification key.
- **~/.ssh/id_rsa.pub**: It is a public key.

6. Gerrit – Add Your SSH Key

You can add SSH key to the ssh-agent on different platforms discussed further.

Linux

Use the following command on Linux system to add SSH key.

```
cat /home//.ssh/id_rsa.pub
```

Windows

Open the GIT GUI and go to **Help -> Show SSH Key** as shown in the following image.



Then, click the **Copy To Clipboard** button, to copy the key to the clipboard.



Mac

In Mac OS X, you can copy ***id_rsa.pub*** contents to the clipboard using the following command.

```
$ pbcopy < ~/.ssh/id_rsa.pub
```

7. Gerrit – Add SSH Key to Your Gerrit Account

SSH key can be added to the Gerrit account using the following steps:

Step 1: First, create an account at wmflabs.org services.

Username (help me choose)

Password

Confirm password

Email address

Instance shell account name

You're required to choose a shell account name. It must start with a-z, and can only contain lowercase a-z, 0-9, and - characters.

- If you already had a [SVN account](#) with the same username, [ask it to be created manually](#) and include [this information](#).
- If that's not the case, ask someone to verify from a labs instance whether the shell username is taken, by issuing the command `groups $username` on a shell.

Create your account

Step 2: Next, sign in to the [web interface for Gerrit](https://gerrit.wikimedia.org/r/login/%23%2Fq%2Fstatus%3Aopen).

Step 3: Then in the top right corner, click your username and select the Settings option.



Here, we have created an account with the name John to make use of Gerrit.

Step 4: Click the "SSH Public keys" option on the left-side menu and paste the SSH Public key in the field.

8. Gerrit – Add SSH Key to Use with Git

You can add SSH key to Git using the following commands:

Step 1: Open Git Bash and get the ssh-agent using the following command.

```
$ eval 'ssh-agent'
```

Step 2: Next, add the SSH key to the ssh-agent using the following command.

```
$ ssh-add ~/.ssh/id_rsa
```

Step 3: Now, run the ssh using the following command, which matches the SSH fingerprint used when logging for the first time.

```
$ ssh -p 29418 @gerrit.wikimedia.org
```

```
Admin@Admin-PC MINGW32 ~
$ ssh -p 29418 xyz123@gerrit.wikimedia.org

****      Welcome to Gerrit Code Review      ****

Hi Abc123, you have successfully connected over SSH.

Unfortunately, interactive shells are disabled.
To clone a hosted Git repository, use:

git clone ssh://xyz123@gerrit.wikimedia.org:29418/REPOSITORY_NAME.git

Connection to gerrit.wikimedia.org closed.
```

In the above screenshot, you can see that **xyz123** is a instance shell account name, which is used while creating Gerrit account and **Abc123** is a user name of your Gerrit account.

9. Gerrit – Download Examples Using Git

You can download the example using Git along with the source code of any project organized at **gerrit.wikimedia.org** using the following Git Bash command.

```
$ git clone  
ssh://<user_name>@gerrit.wikimedia.org:29418/mediawiki/extensions/examples
```

The *git clone* command clones a directory into a new directory; in other words gets a copy of an existing repository. When you run the above command, you will get a screenshot similar to the following.

```
Admin@Admin-PC MINGW32 ~  
$ git clone ssh://xyz123@gerrit.wikimedia.org:29418/mediawiki/extensions/example  
s  
Cloning into 'examples'...  
remote: Total 952 (delta 0), reused 952 (delta 0)  
Receiving objects: 100% (952/952), 147.16 KiB | 132.00 KiB/s, done.  
Resolving deltas: 100% (559/559), done.  
Checking connectivity... done.
```

The above command clones the 'examples' repository and receives the objects, files, etc. from that repository and stores it in your local branch.

Prepare to Work with Gerrit

10. Gerrit – Installing Git-Review

You can work with Gerrit by installing *git-review* on different platforms as discussed in this chapter.

Windows

In Windows, you can install the git-review as listed in the following steps.

Step 1: First, install Python for installing git-review.



Step 2: Keep the Python installation in the default directory (like C:\Python27) instead of installing in any other directory.

Step 3: Next, set the environment variables for Python scripts directory using the path as **C:\Python27\;C:\Python27\Scripts\;**

```
git_review_install
```

Step 4: With version 2.7, Python will install pip automatically. For older version of Python 2.7, you can install pip as described in this [link](#).

Step 5: Run open Git Bash and install the git-review using the following command.

```
$ pip install git-review
```

Linux

In Linux, you can install git-review as described in the following steps:

Step 1: Users of Linux platform do not have root access on shared host. Hence, without root access, you can install git-review locally in user directory using the following commands:

```
virtualenv --python=/usr/bin/python2.6 virtualenv  
virtualenv/bin/pip install git-review==1.21
```

Step 2: You can extend the path to the local binaries using two ways:

```
PATH=$PATH:~/virtualenv/bin/  
PATH=~/virtualenv/bin:$PATH
```

Step 3: Now, use the following command to set up the work with Gerrit.

```
git review -s  
or  
~/virtualenv/bin/git-review -s
```

Step 4: With root access, git-review can be installed using the following command.

```
sudo apt-get install git-review
```

Step 5: If there is no *apt-get* after installing Python, then use the following commands.

```
$ sudo easy_install pip  
$ sudo pip install git-review==1.21
```

Step 6: Run the following command to work with Gerrit.

```
git review -s
```

Mac OS X

In Mac, you can install the git-review using the following steps.

Step 1: Install the Homebrew from this [link](#).

Step 2: Next, install the git-review using the following command.

```
brew install git-review
```

11. Gerrit – Configuring Git-Review

Gerrit is built on top of Git version control system, which extracts the code from other host, pushing changes to the code, submitting the code for review, etc. The default remote name of Git is *origin* and we tell git-review to use this name 'origin' by using the following command.

```
$ git config --global gitreview.remote origin
```

12. Gerrit – Setting Up Git-Review

Git-review can be used to send git branches to Gerrit for reviewing. You can set up git-review using the following command in the project directory.

```
$ git review -s
```

Git-review can be used as the command line tool for configuring Git clone, submitting the branches to Gerrit, fetching the existing files, etc. Git-review looks for the remote named gerrit for working with Gerrit by default.

If git-review finds the Gerrit remote, then it will submit the branch to **HEAD:refs/for/master** at the remote location, and if there is no Gerrit remote access, then git-review looks for the **.gitreview** file at the root of the repository along with the gerrit remote information.

Git-review processes the following internally:

- It will check whether the remote repository works or not for submitting the branches.
- If there is no Gerrit remote access, then it will ask for the username and try to access the repository again.
- It will create a remote access called gerrit that points to Gerrit.
- It will install the commit-msg hook.

How to Submit a Patch

13. Gerrit – Update Master

You can make the master branch up-to-date using the following command. The git-pull command fetches from another local branch or integrates with another repository.

```
git pull origin master
```

```
Admin@Admin-PC MINGW32 ~/examples (master)
$ git pull origin master
From ssh://gerrit.wikimedia.org:29418/mediawiki/extensions/examples
 * branch          master      -> FETCH_HEAD
Already up-to-date.
```

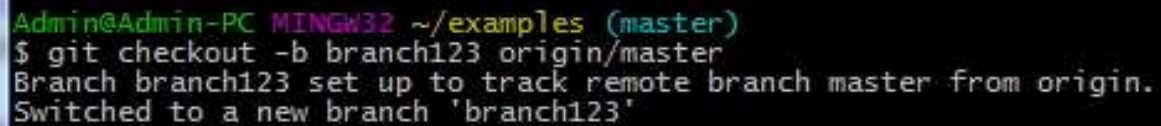
- The command will pull changes from the origin remote (URL of remote to fetch from), master branch, and merge the changes to local checked-out branch.
- The origin master is a cached copy of the last pulled from the origin.
- Git pull is a combination of git fetch (fetches new commits from the remote repository) and git merge (integrates new commits into local branch).
- Git pull merges the local branch with the remote branch by default.

14. Gerrit – Create a Branch

You can create a branch on the local machine using the following command.

```
$ git checkout -b name_of_branch origin/master
```

The above command creates a new branch as shown in the following screenshot.

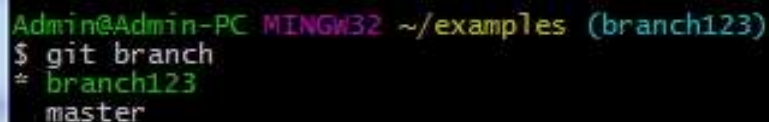


```
Admin@Admin-PC MINGW32 ~/examples (master)
$ git checkout -b branch123 origin/master
Branch branch123 set up to track remote branch master from origin.
Switched to a new branch 'branch123'
```

Here, we have used **branch123** as the new local branch. You can show the new branch from the 'master' using the following command.

```
$ git branch
```

The above command produces the result as shown in the following screenshot.



```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git branch
* branch123
  master
```

Git checkout navigates between the branch, updates the files in the working directory, and informs Git to record the commits on that branch.

15. Gerrit – Make & Commit Your Change

When you modify the code in the local file system, you can check for the changes within the directory using the following command.

```
$ git diff
```

In the project directory, we will modify some changes in the file called **Example/Example.hooks.php** and run the above command. We will get the result as shown in the following screenshot.

```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git diff
diff --git a/Example/Example.hooks.php b/Example/Example.hooks.php
index 0e2b4f5..3cc0f5d 100644
--- a/Example/Example.hooks.php
+++ b/Example/Example.hooks.php
@@ -5,7 +5,7 @@
 * @file
 * @ingroup Extensions
 */
+
class ExampleHooks {
    /**
     * Add welcome module to the load queue of all pages
```

You can check the changes made to the files or the directory using the following command.

```
$ git status
```

```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git status
On branch branch123
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Example/Example.hooks.php

no changes added to commit (use "git add" and/or "git commit -a")
```

The above command allows to see which changes have been staged, which have not, and which files are not tracked by Git.

Next, you can add the changes in the working directory and update the file in the next commit using following command.

```
$ git add Example/Example.hooks.php
```

After adding the file, again run the *git status* command to review the changes added to the staging area as shown in the following screenshot.

```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git status
On branch branch123
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   Example/Example.hooks.php
```

You can see the difference between the index and your last commit, and what contents have been staged, using the following command.

```
$ git diff --cached
```

```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git diff --cached
diff --git a/Example/Example.hooks.php b/Example/Example.hooks.php
index 0e2b4f5..3cc0f5d 100644
--- a/Example/Example.hooks.php
+++ b/Example/Example.hooks.php
@@ -5,7 +5,7 @@
 * @file
 * @ingroup Extensions
 */
+
+class ExampleHooks {
+    /**
+     * Add welcome module to the load queue of all pages
```

You can push the changes to the remote directory from the local repository using the following command.

```
$ git commit
```

When you run the above command, it will ask to add the commit message for your changes. This message will be seen by other people when you push the commit to the other repository.

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch branch123
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#   modified:   Example/Example.hooks.php
#
```

Add the commit message and run the command again as *git commit*, which will display the commit message as shown in the following screenshot.

```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git commit
[branch123 4b400b7] modified version of file
1 file changed, 1 insertion(+), 1 deletion(-)
```

16. Gerrit – Prepare Push Change Set to Gerrit

You need to review the changes in Gerrit before merging them into the master. The changes can be synchronized that have occurred in the master. Use the following command within the branch that you have been working on.

```
$ git pull --rebase origin master
```

- The above command will fetch the changes or commits from the remote branch and rebase the commits on top of the master.
- When you are done with the changes and rebased the commits, you can push your change set to Gerrit for review.
- Git pull --rebase is often used when changes do not deserve a separate branch.
- Git pull is a combination of git fetch and git merge; whereas git pull --rebase is a combination of git fetch and git rebase.

First, run the command as *git pull origin master* as shown in the following screenshot.

```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git pull origin master
From ssh://gerrit.wikimedia.org:29418/mediawiki/extensions/examples
* branch          master      -> FETCH_HEAD
Already up-to-date.
```

Now use the command as *git rebase master* to rebase the commits as shown in the following screenshot.

```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git rebase master
Current branch branch123 is up to date.
```

17. Gerrit – Push Your Change Set to Gerrit

You can submit the patches for review by using the **git-review** command. The change set can be pushed to Gerrit, by running the **git review -R** command as shown in the following screenshot.

```
Admin@Admin-PC MINGW32 ~/examples (branch123)
$ git review -R
remote: Processing changes: new: 1, refs: 1, done
remote:
remote: New Changes:
remote:  https://gerrit.wikimedia.org/r/311409 modified author name
remote:
To ssh://xyz123@gerrit.wikimedia.org:29418/mediawiki/extensions/examples
* [new branch]      HEAD -> refs/publish/master/branch123
```

The **-R** option informs git-review not to complete rebase before submitting git changes to Gerrit.

You can submit the code to other branch rather than the master, using the following command.

```
git review name_of_branch
```

It is also possible to submit the code to a different remote, using the following command.

```
git review -r name_of_remote
```


18. Gerrit – View the Change / Next Steps

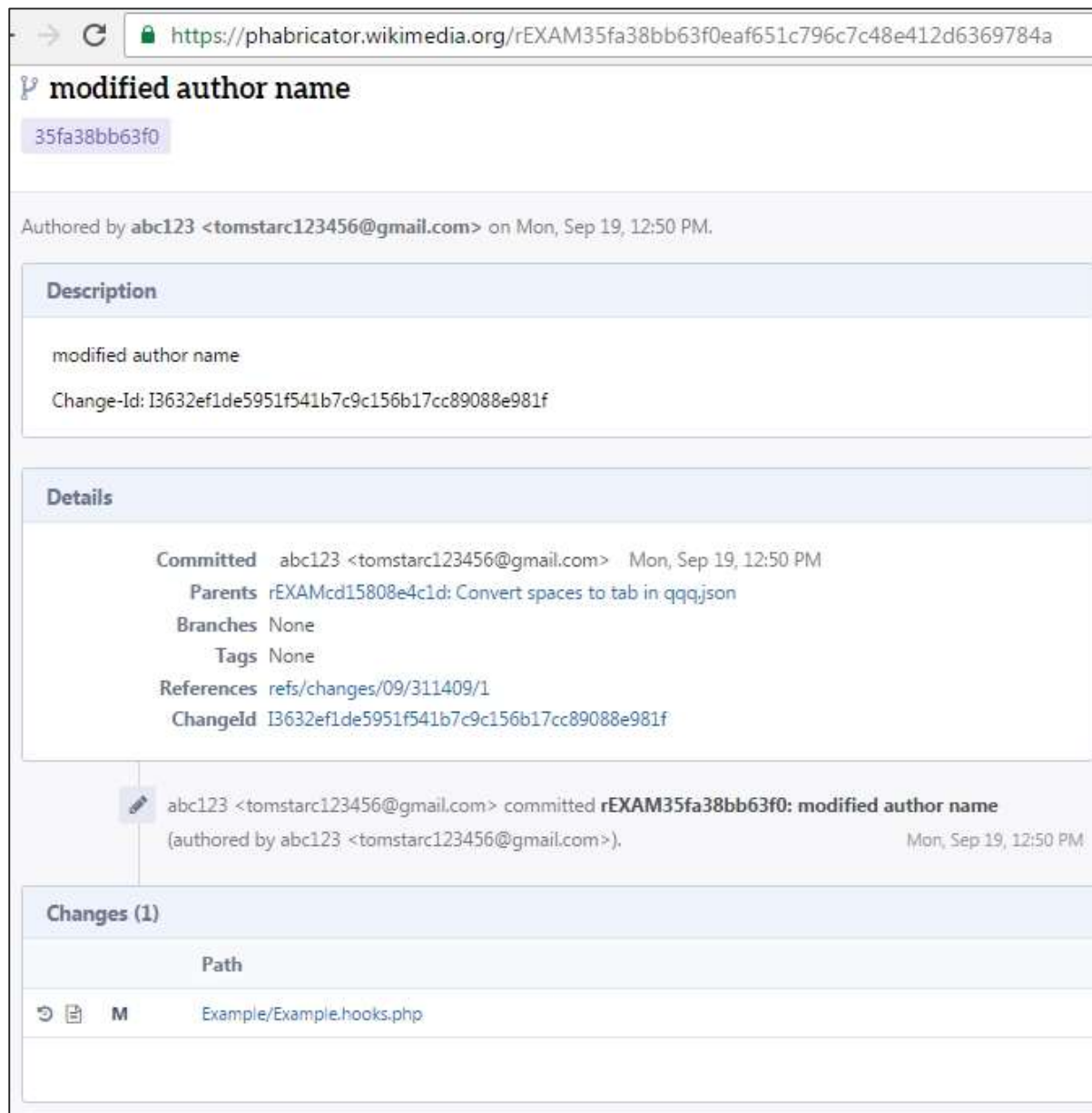
The changes can be viewed in Gerrit dashboard by clicking in this [link](#).

The screenshot shows the Gerrit dashboard for a user. The URL is <https://gerrit.wikimedia.org/r/#/dashboard/self>. The page has a header with the Wikimedia Code Review logo and navigation tabs: All, My (selected), Projects, People, Documentation. Below the header are sub-tabs: Changes, Drafts, Draft Comments, Edits, and Watched Changes. The main section is titled 'My Reviews' and contains three tables: 'Outgoing reviews' with one entry 'modified author name' by 'Abc123' in project 'mediawiki/extensions/examples'; 'Incoming reviews' with '(None)'; and 'Recently closed' with '(None)'.

Click the *modified author name* link, and you will get the following screenshot.

The screenshot shows the detailed view of change 311409. The URL is <https://gerrit.wikimedia.org/r/#/c/311409/>. The page title is 'Change 311409 - Not Code-Review'. The change description is 'modified author name' with 'Change-Id: I3632ef1de5951f541b7c9c156b17cc89088e981f'. On the right, there is a 'Reply...' section with metadata: Owner 'Abc123', Reviewers 'Aklapper' and 'jenkins-bot', Project 'mediawiki/extensions/examples', Branch 'master', Topic 'branch123', Strategy 'Merge if Necessary', and Updated '15 hours ago'. Below this are buttons for 'Cherry Pick', 'Rebase', 'Abandon', and 'Follow-Up'. A 'Code-Review' section shows '-2' by 'Aklapper' and '+1' by 'jenkins-bot'. The bottom section shows the commit details: Author 'abc123', Committer 'abc123', Commit '35fa38bb63f0eaf651c796c7c48e412d6369784a', Parent(s) 'cd15808e4c1d8d0c0d55dfa60aef16e81417159', and Change-Id 'I3632ef1de5951f541b7c9c156b17cc89088e981f'. At the bottom, there is a 'Files' section with a table showing the file path 'Example/Example.hooks.php' and its change size '+1, -1'.

Click the diffusion link to see the changed files with other details as shown in the following screenshot.



The screenshot shows a Gerrit commit page for the commit ID `35fa38bb63f0`. The commit was authored by `abc123 <tomstarc123456@gmail.com>` on Monday, September 19, 2016, at 12:50 PM. The commit message is "modified author name". The change ID is `I3632ef1de5951f541b7c9c156b17cc89088e981f`.

The "Details" section shows the commit was committed by `abc123 <tomstarc123456@gmail.com>` on Monday, September 19, 2016, at 12:50 PM. The parents are `rEXAMcd15808e4c1d: Convert spaces to tab in qqjson`. There are no branches or tags associated with this commit. The references are `refs/changes/09/311409/1`. The change ID is `I3632ef1de5951f541b7c9c156b17cc89088e981f`.

The "Changes (1)" section shows a single change to the file `Example/Example.hooks.php`.

Path
<code>Example/Example.hooks.php</code>

19. Gerrit – Editing via the Web-Interface

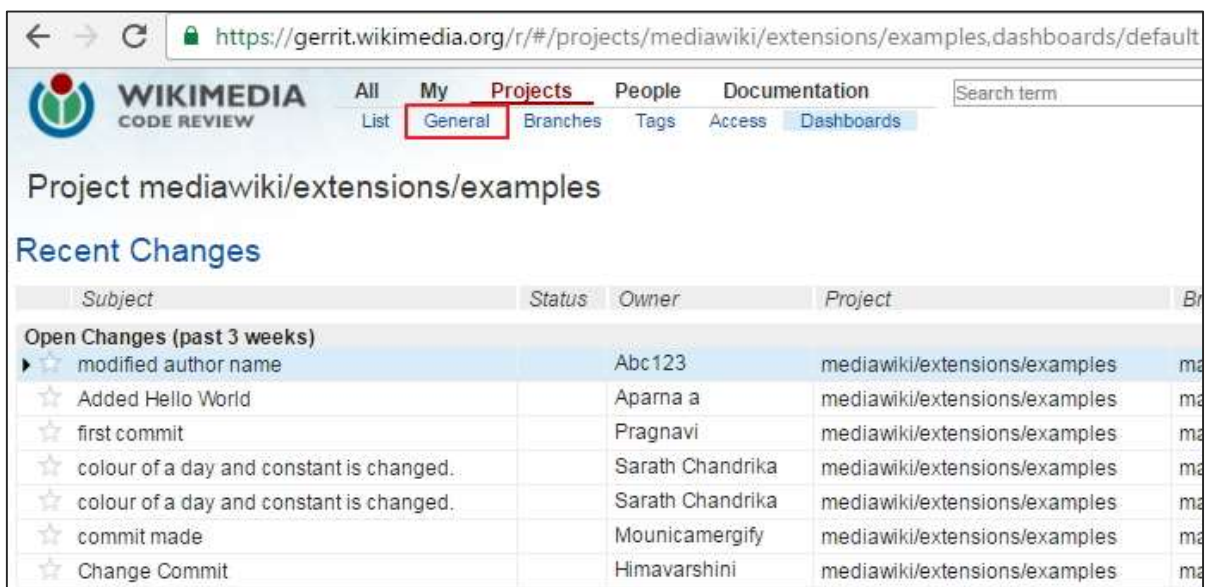
You can edit the project via the web interface after logging in to the Gerrit account as shown in the following steps.

Step 1: Go to Gerrit dashboard by clicking this [link](https://gerrit.wikimedia.org/r/#/dashboard/self). You will get the following screenshot.



Step 2: Next click the **mediawiki/extensions/examples** link specified under *Project* column.

Step 3: Click the *General* link in the toolbar as shown in the following screenshot.



Step 4: When you open the *General* link, it will show a screenshot as the following.

← → ↻ <https://gerrit.wikimedia.org/r/#/admin/projects/mediawiki/extensions/examples>

WIKIMEDIA CODE REVIEW All My Projects People Documentation
List General Branches Tags Access Dashboards Search term

Project mediawiki/extensions/examples

[Clone](#) | [Clone with commit-msg hook](#) | [anonymous http](#) | [http](#) | [ssh](#)

```
git clone https://gerrit.wikimedia.org/r/mediawiki/extensions/examples
```

Description
Examples for how to create an extension. This is not the repo to test out git/gerrit, for that please see the test/* projects.

Project Options
 State:
 Submit Type:
 Allow content merges:
 Create a new change for every commit not in the target branch:
 Require change-id in commit message:
 Maximum Git object size limit:

Contributor Agreements
 Require signed-off-by in commit message:

its-phabricator Plugin
 Enable its-phabricator integration:

Project Commands
 Commands:

Step 5: Click the *Create Change* button and it will open a popup window with some details as shown in the following screenshot.

Create Change

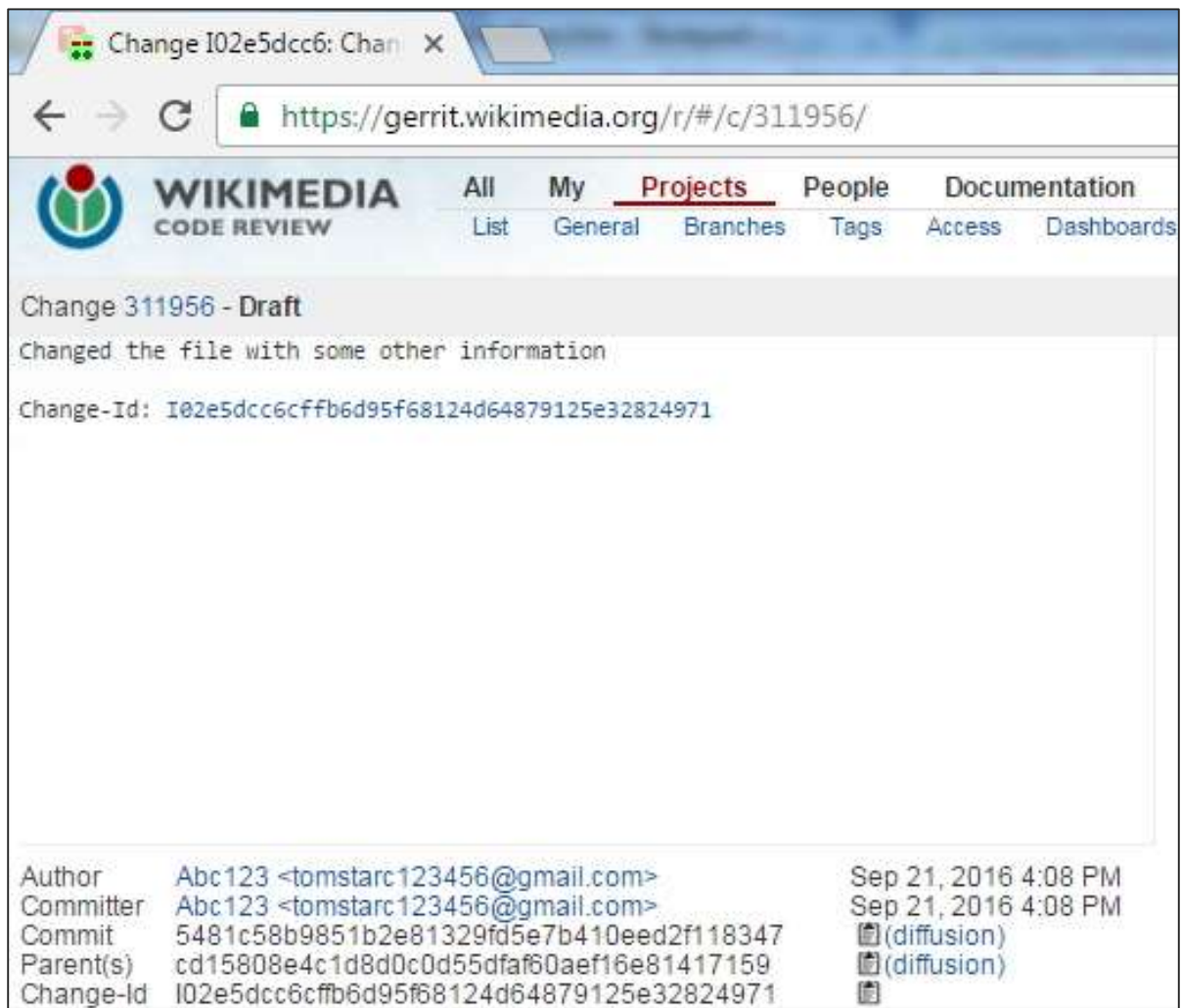
Select branch for new change

Enter topic for new change

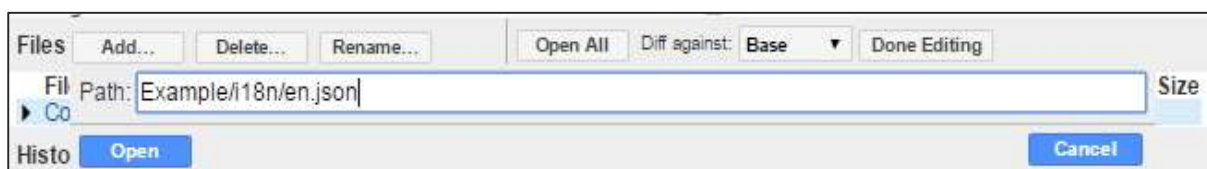
Description
Insert the description of the change.

Step 6: Enter the information and click the *Create* button.

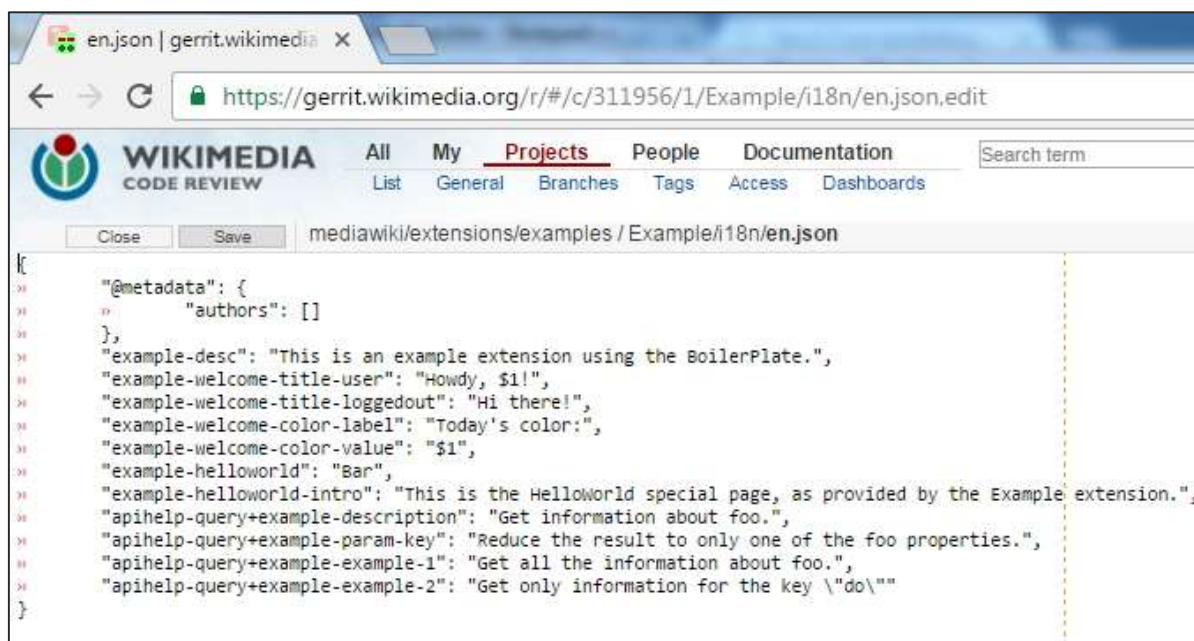
After creating the change, it will display the information as shown in the following screenshot.



Step 7: Click *Edit* and then click *Add*. Now, select the file you want to edit. Here, we have selected the file ***Example/i18n/en.json***.



When you open the file, it will show the json data as specified in the following screenshot.



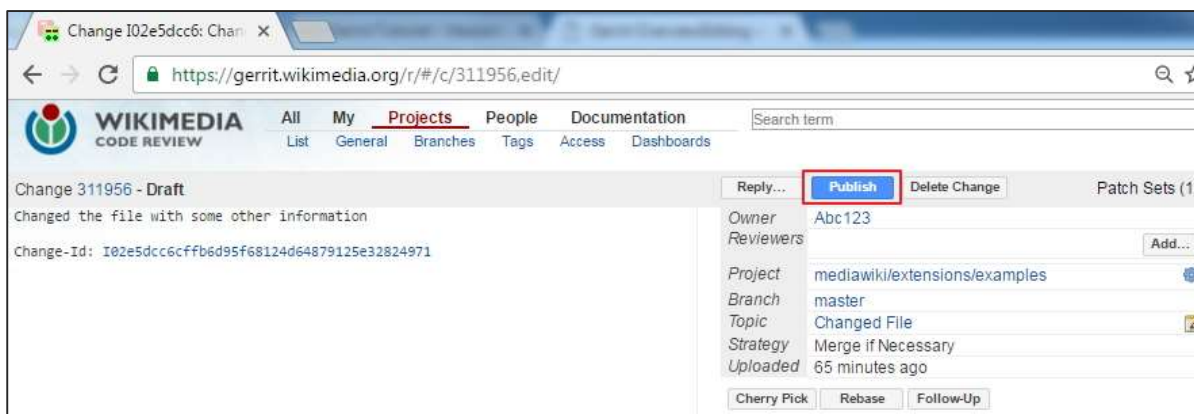
```

{
  "@metadata": {
    "authors": []
  },
  "example-desc": "This is an example extension using the BoilerPlate.",
  "example-welcome-title-user": "Howdy, $1!",
  "example-welcome-title-loggedout": "Hi there!",
  "example-welcome-color-label": "Today's color:",
  "example-welcome-color-value": "$1",
  "example-helloworld": "Bar",
  "example-helloworld-intro": "This is the Helloworld special page, as provided by the Example extension.",
  "apihelp-query+example-description": "Get information about foo.",
  "apihelp-query+example-param-key": "Reduce the result to only one of the foo properties.",
  "apihelp-query+example-example-1": "Get all the information about foo.",
  "apihelp-query+example-example-2": "Get only information for the key \"do\""
}

```

Step 8: Click *Save* and then click the *Close* button.

Step 9: Finally click the *Publish* button to publish the edited file.



Change 311956 - Draft
 Changed the file with some other information
 Change-Id: I02e5dcc6c6cfb6d95f68124d64879125e32824971

Buttons: Reply..., **Publish**, Delete Change

Metadata:

- Owner: Abc123
- Reviewers: Add...
- Project: mediawiki/extensions/examples
- Branch: master
- Topic: Changed File
- Strategy: Merge if Necessary
- Uploaded: 65 minutes ago

Buttons: Cherry Pick, Rebase, Follow-Up

Step 10: You can change commit message by clicking the *Commit Message* link as shown in the following screenshot.



File Path

- Commit Message

History

Abc123	Uploaded patch set 1.
Expand All	

Step 11: Press **e** on the keyboard and add some extra information, if you wish to. Click *Save* and then click the *Close* button.



How Code is Reviewed in Gerrit

20. Gerrit – Review before Merge

Code review is an important part of the workflow in Gerrit. The basic concept is that the code must be reviewed before being merged.

The workflow of the code for [MediaWiki](#) can be reviewed before merging it and also extensions can be reviewed, which customizes the MediaWiki looks and works. There is one special case in which you can push the [internationalization and localization](#) commits.

You can push all the commits to a remote branch when you finish the development. Someone will fetch the changes into local and merge those fetched changes into the local master by creating merge commit. You can push these changes to ***refs/for/master***.

21. Gerrit – Project Owners

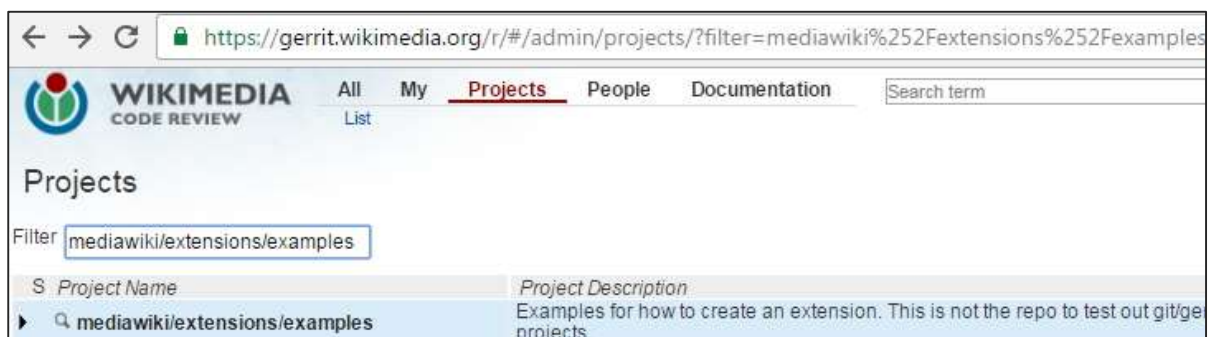
Project owner means that the project belongs to the person mentioned. Project owners is a virtual group in which you cannot add members or other groups in it. The project owner provides access rights to allow permission on the project to different groups.

You can view the access rights of your project using the following steps.

Step 1: Open Gerrit dashboard by clicking this [link](https://gerrit.wikimedia.org/r/#/dashboard/self).



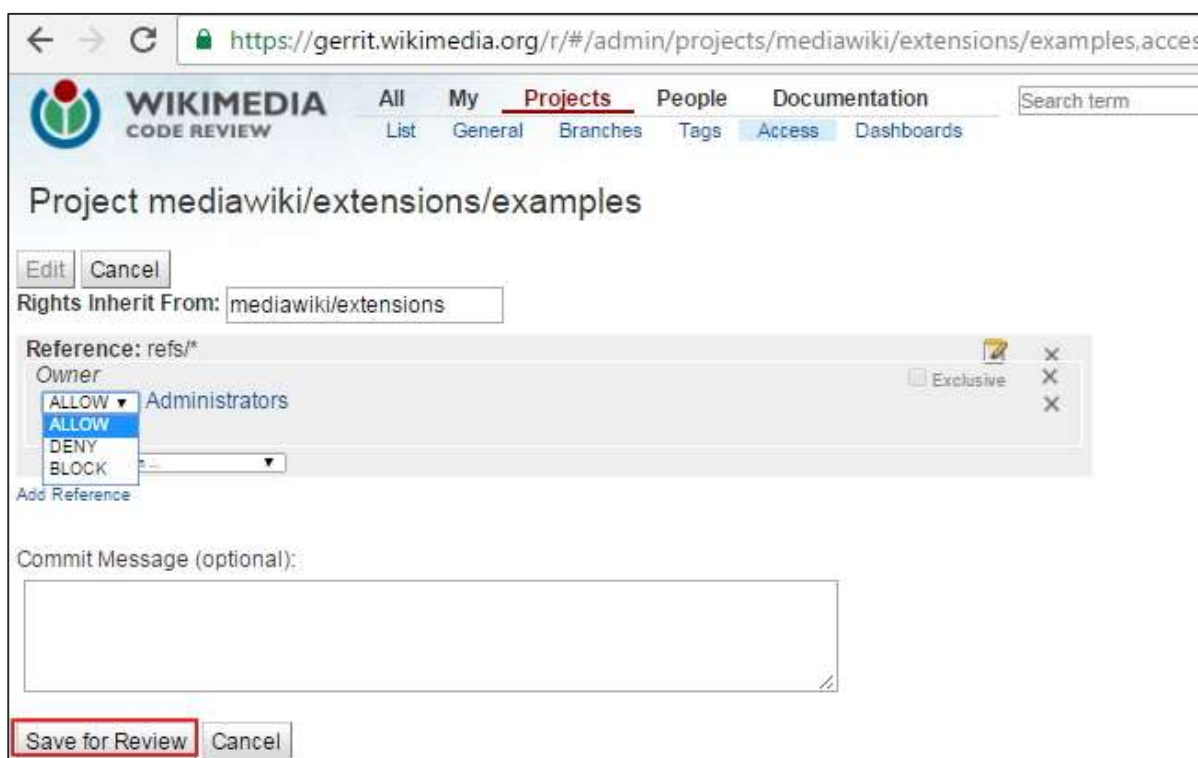
Step 2: Click *Projects* -> *List* option. Search the project in your project list and click it as shown in the following screenshot.



Step 3: When you open your project, click the *Access* option as shown in the following screenshot.



Step 4: Click the *edit* option. You can change the access rights by clicking the dropdown menu. Click the *Save Changes* button as shown in the following screenshot.

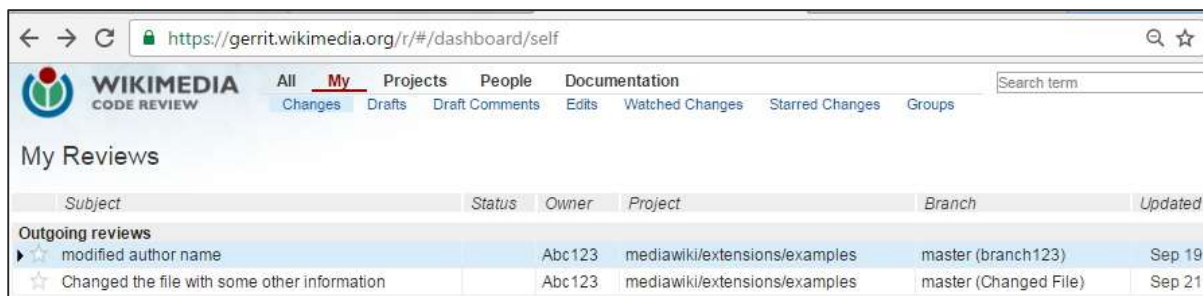


22. Gerrit – How to Comment, Review, and Merge

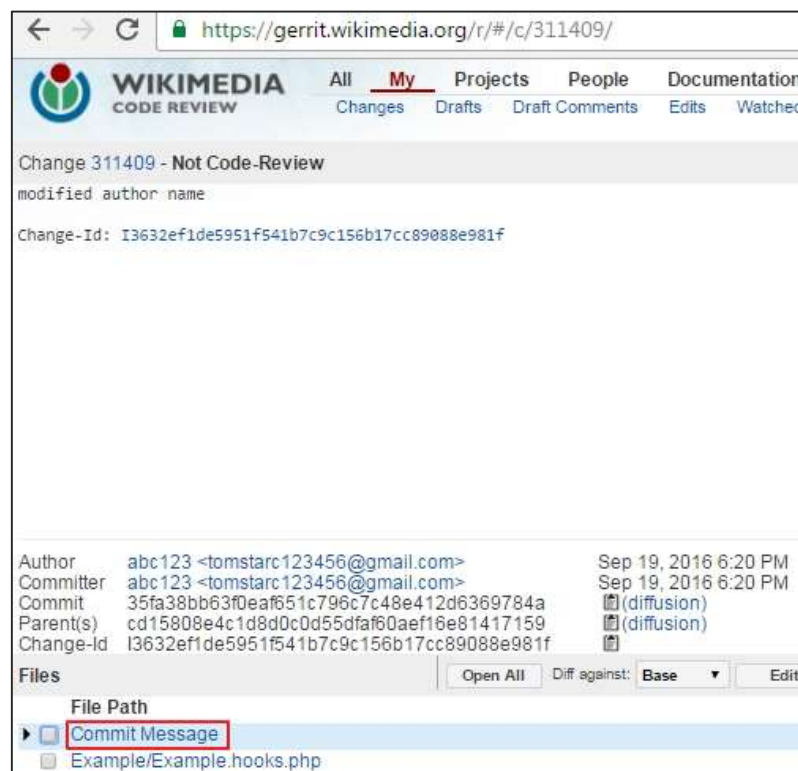
Anyone can review the code and comment on the code in Gerrit. Consider the following steps:

Step 1: Login to Gerrit to open the Gerrit dashboard as specified in the previous chapter.

Step 2: Now, click any subject which contains Gerrit project, branch, updated date, etc. as shown in the following screenshot.



Step 3: Next, it will display a screen. Click the *Commit Message* option as shown in the following screenshot.



The important fields of the change set are like Reviewers, Add Reviewer, Side-by-side off, etc. Comparing patch sets includes selecting the old version history list, expanding the newer patch set details, etc. Reviewing and merging or rejecting the code includes abandon change button, submitting patch button, etc. which are not present in the current version of Gerrit.