Self-Learning Points of Gerrit

Contents

[1. What is Gerrit 4](#_Toc464739893)

[1) Characteristics 4](#_Toc464739894)

[2) Advantage 4](#_Toc464739895)

[3) Disadvantage 4](#_Toc464739896)

[2. Key Points 4](#_Toc464739897)

[1) Push to Gerrit is same as push to Git. Only target branch name is different: **refs/for/** 4](#_Toc464739898)

[2) Difference between new change and new patch set 5](#_Toc464739899)

[=> 5](#_Toc464739900)

[=>A common scenario: 6](#_Toc464739901)

[3) How to revert changes in git 6](#_Toc464739902)

[4) Git has 3 level config 7](#_Toc464739903)

[5) Merge 7](#_Toc464739904)

[6) Rebase 8](#_Toc464739905)

[7) Pull and Push 9](#_Toc464739906)

[8) An access control model in Gerrit 10](#_Toc464739907)

[3. FAQ 13](#_Toc464739908)

[1) What’s SHA1? 13](#_Toc464739909)

[2) How to allow multiple people push code to a same repository? 13](#_Toc464739910)

[*3)* *How to fix Gerrit Code Review error “cannot merge…” “missing dependency”?* 13](#_Toc464739911)

[4. Frequently Used Commands 14](#_Toc464739912)

[Table 1 Difference between Git and Gerrit Push 4](#_Toc464739913)

[Table 2 Difference between Change and Patch Set 5](#_Toc464739914)

[Table 3 Difference of using Soft, Mixed and Hard in git reset B 6](#_Toc464739915)

[Figure 1 Difference between Git and Gerrit Push 5](#_Toc464739916)

[Figure 2 Git has 3 level config 7](#_Toc464739917)

[Figure 3 Before Fast Forward merge 7](#_Toc464739918)

[Figure 4 After Fast Forward merge 7](#_Toc464739919)

[Figure 5 Before Cherry-Pick merge 8](#_Toc464739920)

[Figure 6 After Cherry-Pick merge 8](#_Toc464739921)

[Figure 7 Git before rebase 9](#_Toc464739922)

[Figure 8 Git after rebase: fast forward merge is possible 9](#_Toc464739923)

[Figure 9 An example of access control model in Gerrit 12](#_Toc464739924)

Gerrit

# What is Gerrit

Gerrit is a git server with 2 extra functions: access management and code review.

## Characteristics

TODO

## Advantage

TODO

## Disadvantage

TODO

# Key Points

## Push to Gerrit is same as push to Git. Only target branch name is different: **refs/for/**

|  |  |  |
| --- | --- | --- |
| Push to Git | Push to Gerrit | Notes |
| git push origin HEAD:master  Git:   * Could has 2 commits in 1 push | git push origin HEAD:refs/for/master  Gerrit:   * Create a new branch for the commit you push * Create a new open Gerrit change in Gerrit DB for **each** push: 2 changes |  |

Table Difference between Git and Gerrit Push

Difference between Change and Patch Set in Gerrit

|  |  |
| --- | --- |
| Term | Description |
| Change | The unit of review.  Results in a single commit when merged to the git repository.  Change numbers are unique and never change. |
| Patch Set | A revision of a Change.  Each time a Change is modified, it will receive a new Patch Set.  Patch Set numbering starts from 1.   * Technically, a Patch Set is a unique git commit. |

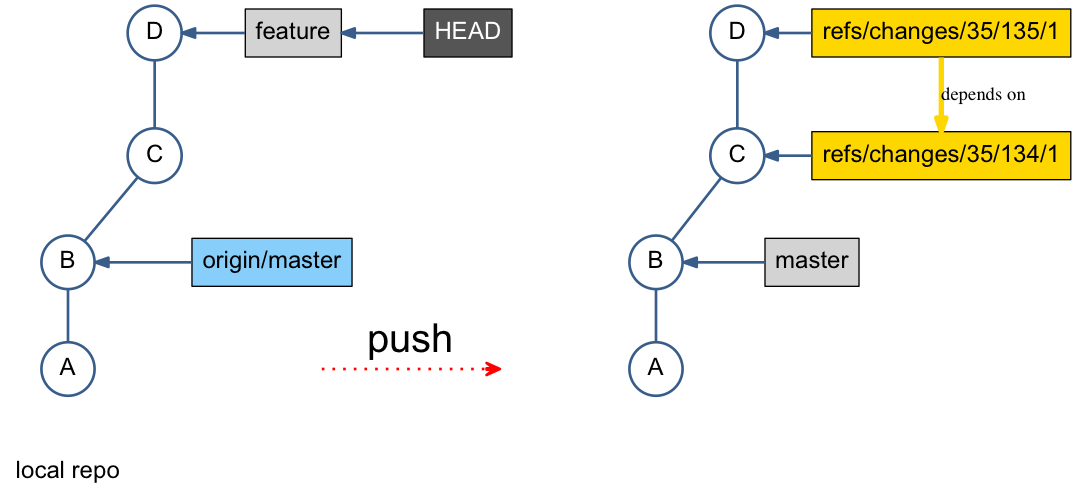


Figure Difference between Git and Gerrit Push

## Difference between new change and new patch set

|  |  |
| --- | --- |
| Change | Patch Set |
| Contains:   * Change-Id * Meta-data (project, owner, etc…) * One or more patch set(s) * Comments * Votes | * New version of an existing change * Only the latest patch set is relevant * No dependencies between patch sets |

Table Difference between Change and Patch Set

=>

1. git commit –amend pushs a new patch set.
2. git fetch downloads an open change locally for test (using commands created by Gerrit.)

=> A common scenario:

Author of Patch Set 1 is not available and somebody else needs to continue and provide Patch Set 2

1. git fetch
2. Create a new branch based on the fetched path set 1
3. Fix the issue
4. git commit –amend
5. push

## How to revert changes in git

* When change is only in working directory, not staged yet.

TODO

* When change is staged, not committed yet.

TODO

* When change is committed.

TODO

* Difference of using Soft, Mixed and Hard in git reset B

|  |  |  |  |
| --- | --- | --- | --- |
| Reset | Working Directory | Stage (Index) | Remote Repo(Branch) |
| soft | No | No | Yes |
| mixed | No | Yes | Yes |
| hard | Yes | Yes | Yes |

Table Difference of using Soft, Mixed and Hard in git reset B

## Git has 3 level config

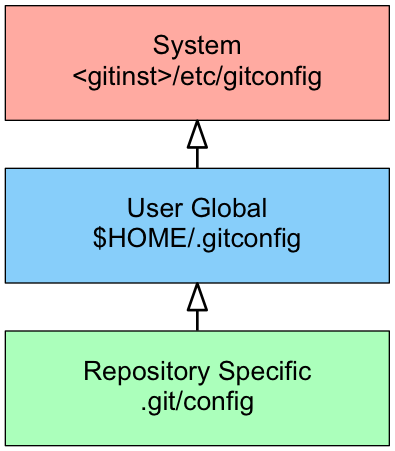


Figure Git has 3 level config

## Merge

* Merge commit is a commit with more than one parent.
* git merge feature1 by default is Fast Forward merge, which just moves the pointer, no new merge commit is created.

<= this is why merge in git is fast.

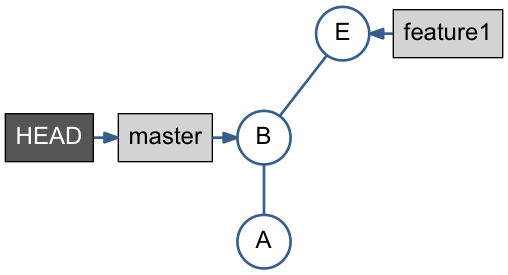


Figure Before Fast Forward merge

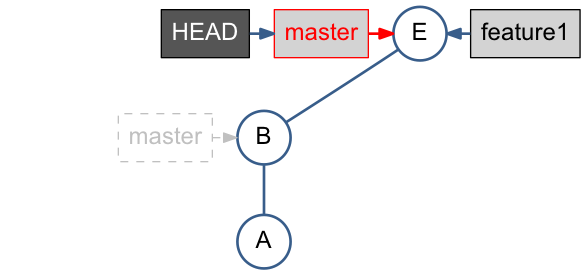


Figure After Fast Forward merge

* git cherry-pick feature1 applies only changes done by F, means the diff-2, has no parent relation to F.

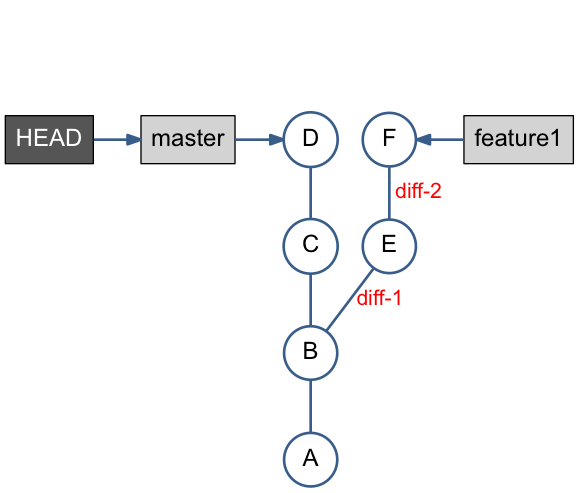
**

Figure Before Cherry-Pick merge

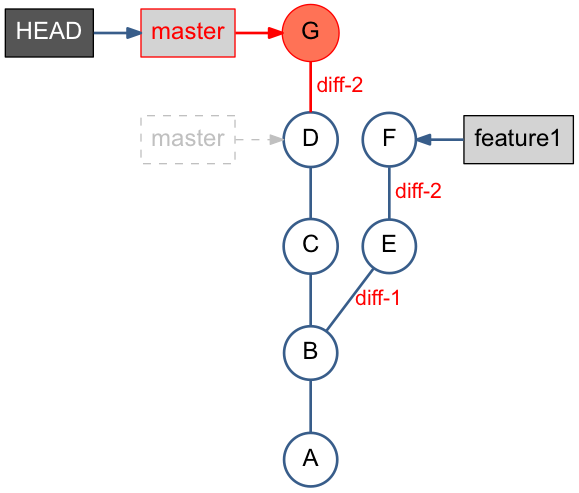
**

Figure After Cherry-Pick merge

## Rebase

* **重新定义某个分支的参考基准.** 就好比移花接木那樣（稼接），把某個樹枝接到別的樹枝。
* git rebase master redo the work done in teature1 branch on top of the master*.*
* **Alternative to Merge** – Keeping history linear
* Fast Forward merge is possible after rebase.
* git rebase –onto *<new* base *commit> <current base commit>指定要從哪裡開始接枝*

[**https://blog.yorkxin.org/posts/2011/07/29/git-rebase/**](https://blog.yorkxin.org/posts/2011/07/29/git-rebase/)

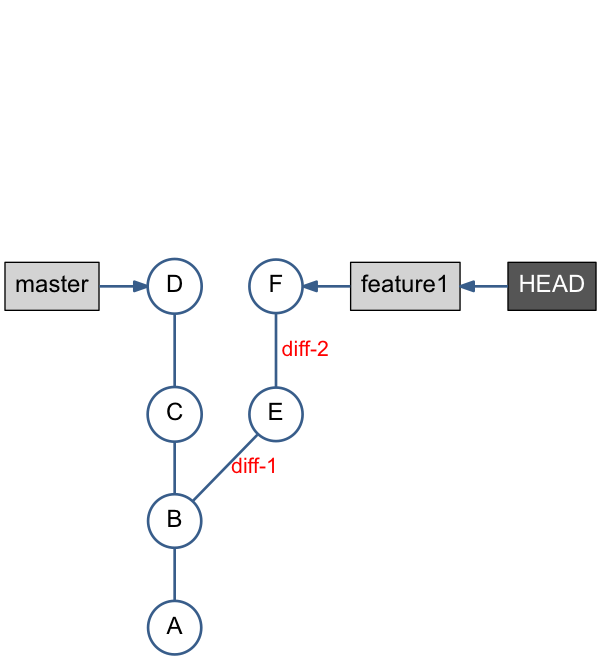
****

Figure Git before rebase

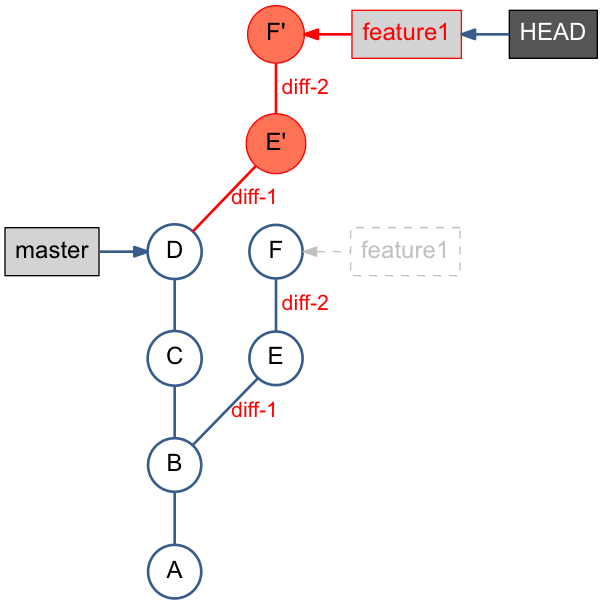
****

Figure Git after rebase: fast forward merge is possible

## Pull and Push

Pull is equal to 3 possibilities:

* git pull = git fetch + git merge

or

* git pull = git fetch + git rebase

or

* git pull = git fetch

Push could mean to deal with 3 scenarios:

* push (when remote ‘origin’ repo is **NOT** changed)

or

* fetch, merge, push (when remote ‘origin’ repo is changed)

or

* fetch, rebase, push (when remote ‘origin’ repo is changed)

## An access control model in Gerrit

All-Projects ->

*Global Capabilities, such as Access Database, Administrate Server, Create Group, Create Project, Delete Own Project, Email Reviewers, Delete Own Images, JCWB Admin, Create Service User, Stream Events, View Caches, View Connections, View Plugins, View Queue, etc. (for Admin)*

*Reference: ^refs/(?!tags).\*$*

*Reference: refs/\**

*Reference: refs/meta/\**

*Reference: refs/meta/config*

*Reference: refs/for/refs/meta/config*

*Reference: refs/tags/\**

->Default Settings ->

*Reference: refs/\**

*Reference: refs/heads/\**

*Reference: refs/for/refs/*

*Reference: refs/tags/\**

*Reference: refs/heads/stages/\* -> Push: ALLOW: xmakeDeleteBranch.*

->${Project1} ->

*Reference: refs/\**

*Label Code-Review (for Dev)*

*Label Verified (for Dev)*

*Submit (for Dev)*

*…*

*Create Reference (for Admin)*

*Push (for Admin)*

*…*

*Reference: refs/heads/\**

*Reference: refs/for/refs/*

*Reference: refs/tags/\**

All-Projects ->

*Global Capabilities, such as Access Database, Administrate Server, Create Group, Create Project, Delete Own Project, Email Reviewers, Delete Own Images, JCWB Admin, Create Service User, Stream Events, View Caches, View Connections, View Plugins, View Queue, etc. (for Admin)*

*Reference: ^refs/(?!tags).\*$*

*Reference: refs/\**

*Reference: refs/meta/\**

*Reference: refs/meta/config*

*Reference: refs/for/refs/meta/config*

*Reference: refs/tags/\**

->Default Settings ->

*Reference: refs/\**

*Reference: refs/heads/\**

*Reference: refs/for/refs/*

*Reference: refs/tags/\**

*Reference: refs/heads/stages/\* -> Push: ALLOW: xmakeDeleteBranch.*

->${Project1} ->

*Reference: refs/\**

*Label Code-Review (for Dev)*

*Label Verified (for Dev)*

*Submit (for Dev)*

*…*

*Create Reference (for Admin)*

*Push (for Admin)*

*…*

*Reference: refs/heads/\**

*Reference: refs/for/refs/*

*Reference: refs/tags/\**

Figure 9 An example of access control model in Gerrit

# FAQ

### What’s SHA1?

SHA1 is a globally unique commit ID

SHA1 is a function of the commit object content.

SHA1 is a 40-digit hexadecimal number, seen in git log output, git history view etc. ex. e168254… 23400e6…

### How to allow multiple people push code to a same repository?

Using core.sharedRepository to control access.

Ref: <http://criticallog.thornet.net/2010/01/07/sharing-your-git-repository/>

Steps:

* Create a group git, put multiple people into it.

useradd git

usermod -g git steven

useradd -g git nicholas

chmod 770 /home/git

chmod g+w objects/

* *Update .git/config, adding sharedRepository*

sharedRepository=1

加这个参数的目的是git在objects目录下创建的目录的属性由

drwxr-xr-x 变成 drwxrwsr-x

* Done.

### *How to fix Gerrit Code Review error “cannot merge…” “missing dependency”?*

Error msg:

Code Review error “cannot merge…”

missing dependency.

It happened after Code-Review+2, Verified+1, when clicking Submit.

Root-cause:

1. No change is submitted after this change was published.
2. If msg is as below:

*Change could not be merged because of a missing dependency.*

*The following changes must also be submitted:*

other changes are not submitted yet.

Ref: <https://git.eclipse.org/r/#/c/58169/>

Fix:

1. Rebase

Click “Change parent revision”

-> input change number of the fore-submitted change

-> Rebase.

1. Submit other changes, then re-try
2. [Need validate] Cherry-pick

# Frequently Used Commands

TODO