Merlot Gerrit Vendor User Guide

Gerrit is a code review system that we use to give external partners access to a read-only mirror of the MOS codebase and allow them to submit changes for us to review and merge. This guide will show you how to get started with Gerrit and walk you through some typical workflows.

## Create Gerrit Account

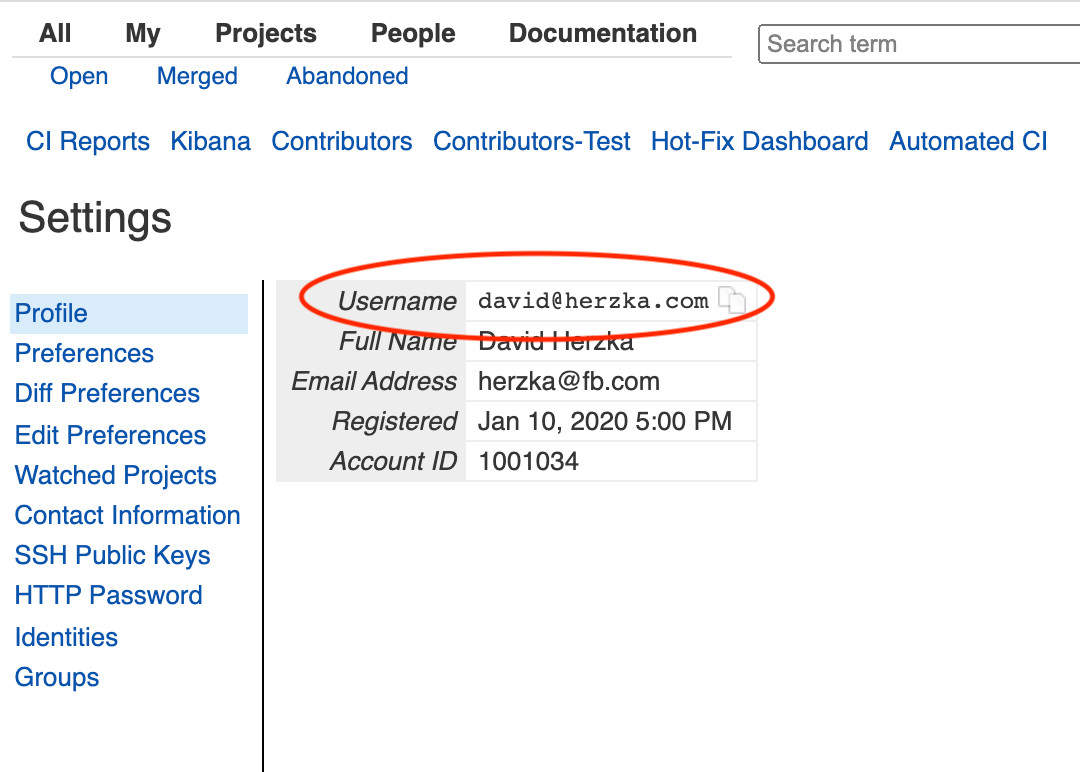
1. You will be using a Facebook account to log in. **The account’s primary email must be your work email address.** Either create a new Facebook account using your work email address or, if you already have an account, change the primary email to your work email.
2. Check if you can ping [agerrit.building8.co](http://agerrit.building8.com/)m. If you cannot, your IP is not on the server’s allowlist.
3. Log into the Facebook account you will use to log into Gerrit and submit an access request using this form: <https://www.facebook.com/help/contact/366872637830604>
4. In order to be allowed access, you will be made a tester of the Building8-Gerrit Facebook app. Once you are notified via email that this has been done, you must accept the role on this page: <https://developers.facebook.com/requests>
5. Log into Gerrit: [https://agerrit.building8.com](https://agerrit.building8.com/). This will create the Gerrit account.
6. After you’ve logged in, let us know by replying to the confirmation email from step 4.
7. Wait for confirmation that you’ve been granted access to the MOS repo.

## Authenticate with Gerrit

1. Generate an SSH key pair:

|  |
| --- |
| ssh-keygen -f ~/.ssh/id\_rsa\_gerrit |

1. Log into Gerrit: [https://agerrit.building8.com](https://agerrit.building8.com/)
2. On the Gerrit settings page, add your SSH public key by pressing “Add Key…” and pasting the contents of ~/.ssh/id\_rsa\_gerrit.pub
3. Get your Gerrit username from your user profile: [https://agerrit.building8.com/#/settings/](https://l.workplace.com/l.php?u=https%3A%2F%2Fagerrit.building8.com%2F%23%2Fsettings%2F&h=AT2MVmGRRxRoa7Ypr8VAODOq8PxkNLw5xKbZXlGBF2xf1oBdKQ1ynwmY6kRtYAtlgXQg0Hw61nq7HI9yTPrrrJ7uL34ZoGL1513rR563sHhX1Epn22TV1g6JA94UkIwOnYkDzUA1qPYtWbdKzty1zL2FHZdqodh1-_CNQA)



1. Add the following to $HOME/.ssh/config:

|  |
| --- |
| Host agerrit.building8.com  User <your Gerrit username>  IdentityFile ~/.ssh/id\_rsa\_gerrit  Port 29418 |

1. Test your connection:

|  |
| --- |
| ssh <your Gerrit username> @agerrit.building8.com |

This should print out a message similar to:

|  |
| --- |
| \*\*\*\* Welcome to Gerrit Code Review \*\*\*\*     Hi <userId>, you have successfully connected over SSH.     Unfortunately, interactive shells are disabled.   To clone a hosted Git repository, use:     git clone ssh://<userid>@agerrit.building8.com:29418/REPOSITORY\_NAME.git    Connection to agerrit.building8.com closed. |

## Clone the Repo

1. Install repo (These commands assume ~/bin exists and is on your path)

|  |
| --- |
| curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo chmod a+x ~/bin/repo |

1. Create and enter a directory for the repo:

|  |
| --- |
| mkdir ~/mos cd ~/mos |

1. Initialize the repo and sync it for the first time:

|  |
| --- |
| repo init -u ssh://agerrit.building8.com/platform/manifest -b mos-11-eic --no-repo-verify --depth 1 |

The --depth 1 flag will create a shallow checkout with only the latest commit in each project. To get more history, you can increase this value or remove the flag, but the repo will be much larger and take much more time to clone. We recommend using --depth 1, then running `git fetch --unshallow` in specific projects if you need the whole commit history.

1. Sync the repo for the first time

|  |
| --- |
| repo sync -q -j$(nproc) -c |

This combination of flags is better than the default behavior in our experience. `-q` hides a lot of noisy logging that makes it hard to see how far along the sync is. `-j` tells repo to sync multiple projects concurrently. `-c` will fetch only the current branch from the server.

1. Install git-lfs (for example, using yum or apt-get)
2. Enable git-lfs:

|  |
| --- |
| git lfs install |

1. Configure git-lfs to get binaries from artifactory:

Add the following to ~/.gitconfig:

|  |
| --- |
| [url "https://aartifactory.building8.com/artifactory/api/lfs/portal"]  insteadOf = https://dewey-lfs.vip.facebook.com/lfs  [credential "https://aartifactory.building8.com"]  username = read-only  helper = "!f() { echo password=AKCp5Z3Bo6UYEgHBwgnkCM3mXfTFhhTLeasmx7ALdfrJnLt97S4hXCAjQDFitb1SA7erFxee4; }; f" |

Pull the latest git lfs binaries:

|  |
| --- |
| repo forall -j$(nproc) -c 'test -e .lfsconfig && git lfs pull' |

## Update the Repo

1. Sync with Gerrit:

|  |
| --- |
| repo sync -q -j$(nproc) -c |

1. Pull the latest git lfs binaries:

|  |
| --- |
| repo forall -j$(nproc) -c 'test -e .lfsconfig && git lfs pull' |

### Make your Code Changes

1. Start a new local development branch:

|  |
| --- |
| repo start *<feature>* . |

Make and test whatever changes you need to and commit them. If you run `git show`, your commit message should have a “Change-Id: …” line added to the bottom automatically. If it doesn’t, something may have gone wrong during the clone process.

|  |
| --- |
| git commit -s |

### Porting Existing Changes

If the change is port of existing work, it is important to retain the original history. Changes should be cherry-picked, and there is an extra flag in the commit command.

|  |
| --- |
| git commit -x -s |

By including -x, this will include the SHA from the original cherry pick.

If there are conflicts or other issues that must be resolved in the cherry-pick, those must be clearly called out in the commit message after amending the commit.

### Commit Message Format

Facebook uses a consistent Commit message format not only for communication and tracking, but to help with automation. Each commit message shall have the following format, and each section should be separated by a blank line:

* Commit title (67 characters max)
* “Summary:” section giving details of changes
* “Test Plan:" section with details on testing
* line "Signed-off-by: {user}"
  + This should be added by the -s flag when committing

#### Sample

|  |
| --- |
| Add null check to avoid crash in TelephonyService  Summary:  When scenario X occurs, the Y pointer is null. Add in a check to handle the exception.  Test Plan:  Scenario X step A  Scenario X step B  Scenario X step C  Ensure exception is handled properly  Change-Id: Ief6663fa1e95b3612a55e0de18f9562853ba6409  Signed-off-by: Christian Flowers <flowers@fb.com> |

### Code Formatting

After committing your code but before pushing to gerrit, you must run lint tools to ensure standards are met. C/C++ uses clang for local formatting, with the clang-format definition checked into the repository already.

|  |
| --- |
| git [vendor/qcom/proprietary/llvm-arm-toolchain-ship/4.0/bin/git-clang-format](https://www.internalfb.com/intern/aosp/browse/?corpus=aosp-mos&repo=aosp%2Fplatform%2Fvendor%2Fqcom%2Fproprietary&branch=mos&path=llvm-arm-toolchain-ship%2F4.0%2Fbin%2Fgit-clang-format&diffusion_app_token=browse) HEAD^ --style=file |

For java code, [google-java-format](https://github.com/google/google-java-format) 1.7 shall be used.

|  |
| --- |
| java -jar vendor/oculus/arcanist/lint/bin/google-java-format-1.7-all-deps.jar –aosp –replace <files> |

### Copyright Header

All new source files must contain this copyright header:

|  |
| --- |
| // (c) Facebook, Inc. and its affiliates. Confidential and proprietary. |

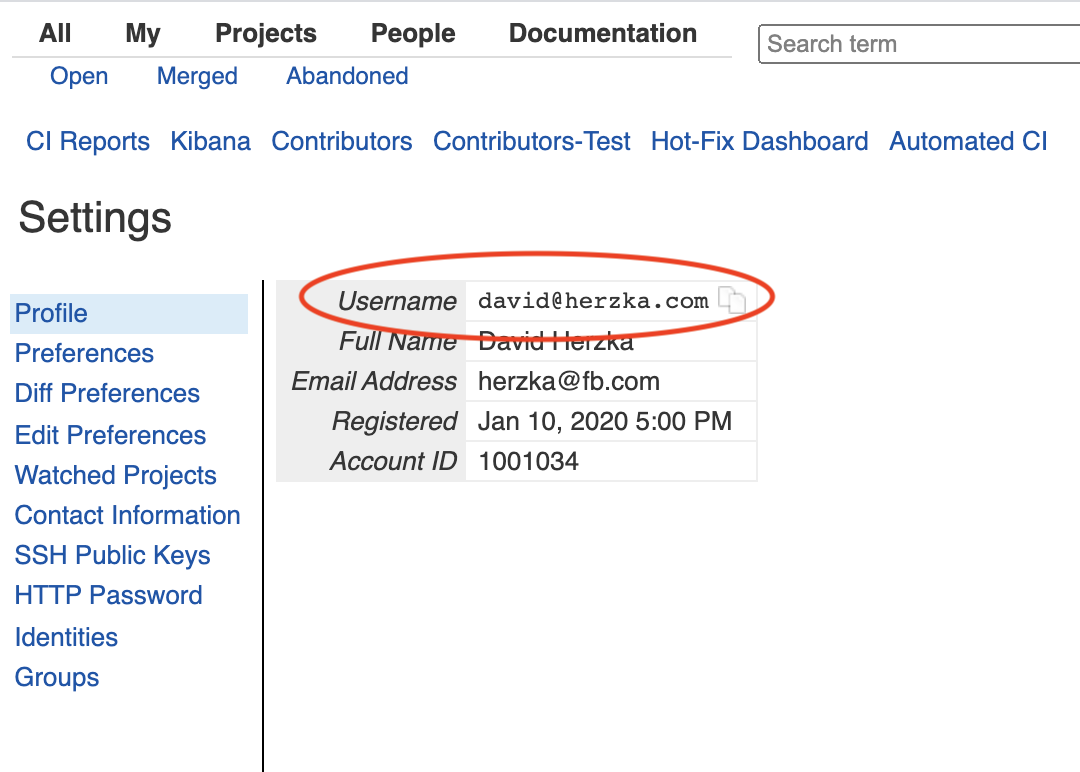
For existing or ported files, existing copyright headers must be retained.

## Upload a Change for Review

`cd` into the project you would like to change

1. Make sure git is configured with your gerrit username

Get your Gerrit username from your user profile:[https://agerrit.building8.com/#/settings/](https://agerrit.building8.com/" \l "/settings/)



Add this username to your git config:

|  |
| --- |
| git config --global review."https://agerrit.building8.com".username <username> |

1. Upload the change:

|  |
| --- |
| repo upload . |

If this was successful, you will see a message containing the Gerrit URL for the change:

|  |
| --- |
| Upload project docs/source.android.com/ to remote branch mos-11-eic:  branch feature ( 1 commit, Fri Jul 17 16:20:38 2020 -0400):  a5538647 Test change to https://agerrit.building8.com (y/N)? y remote: Processing changes: new: 1, refs: 1, done  remote: remote: New Changes:  remote: https://agerrit.building8.com/35799 Test change  remote: To ssh://agerrit.building8.com:29418/platform/docs/source.android.com  \* [new branch] feature -> refs/for/mos-11-eic  ---------------------------------------------------------------------- [OK ] docs/source.android.com/ feature |

1. Assign the Topic

Changes can be grouped by topics. This is useful because it allows you to easily find related changes by using the topic search operator. Also on the change screen changes with the same topic are displayed so that you can easily navigate between them.

Often changes that together implement a feature or a user story are group by a topic.

Assigning a topic to a change can be done in Gerrit. A change must have a topic to be imported into Facebook infrastructure for code review, so please make sure that any change that is ready to be reviewed and merged has a topic. Topic names must not contain spaces.

Graphical user interface, text, application

Description automatically generated

## Fetching a Patch

If the original repo is no longer available, or another developer wishes to use or fix a patch, it can be easily retrieved from gerrit.  Open the change, and under Download and copy the command for Checkout.  You can then run this from your repo.

|  |
| --- |
| // (checkout command copied from gerrit change screen)   $ git fetch https://gerrithost/myProject refs/changes/74/67374/2 && git checkout FETCH\_HEAD |

Graphical user interface, text, application

Description automatically generated

## Pushing Patch Updates

After changing your patch, you need to re-push for further review.

1. Make and test your changes.
2. Amend your existing commit. Be sure to retain the change-id.

|  |
| --- |
| $ git commit –amend |

1. Push your changes back to Gerrit.

|  |
| --- |
| $ repo upload . |

## Patch acceptance criteria

For a patch to be considered ready-for-merging by continuous Integration, it must pass the following criteria:

1. Commit message must consist of a short one-line head, blank line, and commit message (this is the standard kernel and Android practice).
2. Each commit message must include a "Testing:" paragraph describing how the author has validated his change.
3. Each gerrit change must have a short topic string set by the author. Topic is how we group related changes that span across different projects, and submit them together.
4. Change must have at least one CR+2 from a project owner (Gerrit +2 code review score).
5. Change should have at least one CR+1 from a peer.
6. Patch author must have marked his change with V+1 (Verified), in order to signal his work as ready for review and submission.

Only after these steps are done, especially the CR+1 and Verified+1, will the Merlot team begin internal inspections.  Gerrit should be used for all discussion and review.  After passing the Gerrit review, a Merlot team member shall push the change to our internal system.  Once final approval and automated testing has passed, the change will be included in the internal Merlot repos.

## Final Resolution

After merged internally, the external shared Gerrit repo will be updated with the final change.  Merlot’s CM team is responsible for updating Gerrit to the Abandoned state with the release number.

## Useful Links

Repository browser: <https://agerrit.building8.com/plugins/gitiles/>