# Gerrit Code Review Cheatsheet

A quick reference guide for using Gerrit, a web-based team code collaboration tool. Covers essential commands and workflows for code review and submission.

## Gerrit Workflow Basics

### Basic Workflow Overview

Gerrit enhances the standard Git workflow by introducing a code review stage before changes are merged into the main repository. This ensures higher code quality and adherence to project standards.

The typical workflow involves pushing changes to Gerrit, undergoing review by peers, addressing feedback, and finally, submitting the reviewed and approved changes.

### Key Steps

| | |
|---|---|
| **1. Push Changes:** | Push your changes to Gerrit for review using `git push origin HEAD:refs/for/<branch>`. |
| **2. Code Review:** | Reviewers provide feedback on your changes via the Gerrit web interface. |
| **3. Address Feedback:** | Incorporate reviewer feedback by amending your commit and pushing the updated version to Gerrit. |
| **4. Submit Changes:** | Once your changes have received the required approvals, submit them through the Gerrit web interface to be merged into the main branch. |

### Gerrit Web Interface

The Gerrit web interface is where you'll spend most of your time. It allows you to view changes, provide reviews, add comments, and submit changes.

Key features include:

- **Dashboard:** Overview of your changes and reviews assigned to you.
- **Change View:** Detailed view of a specific change, including diffs, comments, and review status.
- **Search:** Powerful search functionality to find changes based on various criteria.

## Essential Git Commands for Gerrit

### Pushing Changes for Review

The primary way to submit code for review is through the `git push` command with a specific refspec.

```
git push origin HEAD:refs/for/<branch>
```

Pushes the current commit to Gerrit for review on the specified branch.

**Example:**

`git push origin HEAD:refs/for/master` - Pushes the current commit for review on the `master` branch.

### Amending Commits

To update a commit based on review feedback, you need to amend the existing commit and push it again. Gerrit recognizes amended commits by their Change-Id.

```
git commit --amend
```

Opens the commit message editor to modify the commit. Make sure to keep the `Change-Id` line intact.

```
git push origin HEAD:refs/for/<branch>
```

After amending, push the updated commit to Gerrit.

### Fetching Changes

You may need to fetch changes from Gerrit to test them locally before providing a review.

```
git fetch origin refs/changes/<X>/<Y>/<Z>
```

Fetches a specific change from Gerrit.

- `<X>`: Last two digits of the change number.
- `<Y>`: The change number.
- `<Z>`: Patch set number.

```
git checkout FETCH_HEAD
```

Checks out the fetched change.

## Gerrit Specifics

### Change-Id

Gerrit uses a `Change-Id` to track revisions of a change. This ID is automatically added to the commit message when you first push a new change.

**Important:** Do not modify or remove the `Change-Id` line when amending commits. Doing so will create a new change in Gerrit instead of updating the existing one.

If you accidentally create a commit without a Change-Id (e.g., when committing directly on the command line), you can use the `git commit --amend` command to add it. Gerrit provides a hook to automatically generate a Change-Id.

### Reviewer Actions

As a reviewer, you can perform several actions on a change through the Gerrit web interface:

- **Approve/Reject:** Indicate whether the change meets the project's quality standards.
- **Add Comments:** Provide specific feedback on lines of code or the overall change.
- **Add Reviewers:** Request reviews from other team members.

- **Verify:** Indicate that change builds and passes tests.
- **Submit:** Submit the change after it receives the required approvals. (Typically, only users with submit permissions can perform this action.)

### Ignoring Files

Sometimes you need to make changes which you do not want to be reviewed, such as debug statements. Gerrit allows these files to be ignored.

Add the file to `.gitattributes` file:

```
<file_name> gerrit-upload:false
```

## Advanced Gerrit Features

### Cherry-Picking Changes

You can cherry-pick changes from Gerrit to apply them to a different branch.

```
git cherry-pick <commit-hash>
```

Cherry-picks the specified commit. You can find the commit hash in the Gerrit web interface.

After cherry-picking, you may need to resolve conflicts and then push the changes to Gerrit for review on the target branch.

### Rebase Workflow

Sometimes, changes will have merge conflicts after another change is submitted to the same branch. Using rebase allows you to update your current change with the latest code.

```
git rebase origin/<branch>
```

Rebases the commits from the remote branch into your current branch.

Push this change to Gerrit to update for review.

## Gerrit Permissions

Gerrit uses a powerful permission system to control access to projects and branches. Permissions can be assigned to users or groups.

Common permissions include:

- Read: Allows users to view the project.
- Submit: Allows users to submit changes.
- Code Review: Allows users to provide code reviews.
- Administrate: Allows users to manage the project and its permissions.

Consult your Gerrit administrator for details on the permission setup for your project.