Gerrit Mastery Guide: The Essential 20% for 80% Results

🎯 What You'll Master

This guide focuses on the **most impactful Gerrit concepts** that will make you productive immediately:

1. **Installation & Setup** - Get Gerrit running quickly
2. **Core Workflow Understanding** - How Gerrit actually works
3. **Change Lifecycle** - From creation to merge
4. **Review Process** - Giving and receiving feedback
5. **Command Line Essentials** - The 5 commands you'll use daily
6. **Common Patterns** - Real-world scenarios you'll encounter

---

🚀 Quick Start: What is Gerrit?

Gerrit in 30 seconds:

- A code review tool built on top of Git
- Every change must be reviewed before merging
- Think of it as a "quality gate" for your code

Key Mental Model:

```
Developer → Gerrit → Reviewers → Merge
    ↓         ↓         ↓         ↓
  Write     Submit    Review   Integrate
   Code     Change    Code     to Main
```

📋 Table of Contents

📚 All Learning Materials Included

This comprehensive guide contains everything you need to master Gerrit:

- ☑ Visual diagrams and flowcharts
- ☑ Real-world examples and scenarios
- ☑ Progressive hands-on exercises
- ☑ Complete command reference
- ☑ Troubleshooting guide
- ☑ Production setup with GitLab integration (private & public)

---

1. Installation & Environment Setup

Option 1: Using Existing Gerrit Server (Recommended for Beginners)

If your organization already has Gerrit set up, you just need to configure your local environment:

Step 1: Install Prerequisites

```
# Install Git (if not already installed)
# Windows: Download from https://git-scm.com/download/win
# macOS: brew install git
# Linux: sudo apt-get install git

# Verify Git installation
git --version
```

Step 2: Configure Git

```
# Set your identity
git config --global user.name "Your Full Name"
git config --global user.email "your.email@company.com"

# Set up SSH key (recommended)
ssh-keygen -t rsa -b 4096 -C "your.email@company.com"
# Add the public key to your Gerrit account
```

Step 3: Clone and Setup Repository

```
# Clone your project repository
git clone ssh://username@gerrit.example.com:29418/your-project
cd your-project

# Install commit-msg hook (CRITICAL!)
```

```
curl -Lo .git/hooks/commit-msg https://gerrit.example.com/tools/hooks/commit-
msg
chmod +x .git/hooks/commit-msg

# Test the setup
git log --oneline -5
```

Option 2: Local Gerrit Setup (For Learning/Testing)

Using Docker (Easiest)

Step 1: Install Docker

```
# Windows: Download Docker Desktop
# macOS: brew install docker
# Linux: sudo apt-get install docker.io docker-compose
```

Step 2: Create Gerrit Container

```
# Create a directory for Gerrit
mkdir gerrit-docker
cd gerrit-docker

# Create docker-compose.yml
cat > docker-compose.yml << 'EOF'
version: '3'
services:
  gerrit:
    image: gerritcodereview/gerrit:latest
    ports:
      - "8080:8080"
      - "29418:29418"
    volumes:
      - gerrit_data:/var/gerrit
    environment:
      - CANONICAL_WEB_URL=http://localhost:8080
    command: |
      sh -c '
        git config -f /var/gerrit/etc/gerrit.config gerrit.canonicalWebUrl
"http://localhost:8080"
        git config -f /var/gerrit/etc/gerrit.config auth.type
"DEVELOPMENT_BECOME_ANY_ACCOUNT"
        /var/gerrit/bin/gerrit.sh run
      '

volumes:
  gerrit_data:
EOF
```

```
# Start Gerrit
docker-compose up -d

# Wait for startup (check logs)
docker-compose logs -f gerrit
```

Step 3: Access Gerrit

- Open browser: http://localhost:8080
- Click "Sign In" → "Become" → Enter your name
- You're now the admin!

Step 4 (Optional): Integrate with GitLab for Testing

You can extend your local Gerrit setup to integrate with GitLab instances for realistic testing scenarios.

Option 2A: Integration with Local GitLab Instance

If you have GitLab running on your local network (e.g., http://gitlab.local:8090):

Step 1: Set up GitLab OAuth Application

```
# In your local GitLab instance:
# 1. Go to Admin Area → Applications (or User Settings → Applications)
# 2. Create new application:
#    - Name: "Local Gerrit Testing"
#    - Redirect URI: http://localhost:8080/oauth
#    - Scopes: read_user, read_api, read_repository
# 3. Note the Application ID and Secret
```

Step 2: Configure Gerrit for GitLab OAuth

```
# Stop Gerrit container
docker-compose down

# Create OAuth plugin directory
mkdir -p gerrit_data/plugins
cd gerrit_data/plugins

# Download GitLab OAuth plugin
wget https://github.com/davido/gerrit-oauth-
provider/releases/download/v3.8.0/gerrit-oauth-provider.jar

# Update docker-compose.yml to include OAuth configuration
cd ..
cat > docker-compose.yml << 'EOF'
version: '3'
```

```
services:
  gerrit:
    image: gerritcodereview/gerrit:latest
    ports:
      - "8080:8080"
      - "29418:29418"
    volumes:
      - gerrit_data:/var/gerrit
      - ./plugins:/var/gerrit/plugins
    environment:
      - CANONICAL_WEB_URL=http://localhost:8080
    command: |
      sh -c '
        git config -f /var/gerrit/etc/gerrit.config gerrit.canonicalWebUrl
"http://localhost:8080"
        git config -f /var/gerrit/etc/gerrit.config auth.type "OAUTH"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.client-id "YOUR_GITLAB_APP_ID"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.client-secret "YOUR_GITLAB_APP_SECRET"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.domain "gitlab.local:8090"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.use-preferred-username "true"
        /var/gerrit/bin/gerrit.sh run
      '

volumes:
  gerrit_data:
EOF

# Replace YOUR_GITLAB_APP_ID and YOUR_GITLAB_APP_SECRET with actual values
# Start Gerrit with OAuth
docker-compose up -d
```

Step 3: Test Local GitLab Integration

```
# Create test repository in local GitLab
# Clone and set up for Gerrit review
git clone http://gitlab.local:8090/your-username/test-project.git
cd test-project

# Add Gerrit remote
git remote add gerrit ssh://your-username@localhost:29418/test-project

# Install commit-msg hook
curl -Lo .git/hooks/commit-msg http://localhost:8080/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg

# Test OAuth login: Go to http://localhost:8080 → Should redirect to GitLab for
auth
```

Option 2B: Integration with GitLab.com

For testing with the public GitLab.com:

Step 1: Create GitLab.com OAuth Application

```
# 1. Go to https://gitlab.com/-/profile/applications
# 2. Create new application:
#    - Name: "Local Gerrit Testing"
#    - Redirect URI: http://localhost:8080/oauth
#    - Scopes: read_user, read_api, read_repository
# 3. Note the Application ID and Secret
```

Step 2: Configure Gerrit for GitLab.com OAuth

```
# Stop current Gerrit
docker-compose down

# Update docker-compose.yml for GitLab.com
cat > docker-compose.yml << 'EOF'
version: '3'
services:
  gerrit:
    image: gerritcodereview/gerrit:latest
    ports:
      - "8080:8080"
      - "29418:29418"
    volumes:
      - gerrit_data:/var/gerrit
      - ./plugins:/var/gerrit/plugins
    environment:
      - CANONICAL_WEB_URL=http://localhost:8080
    command: |
      sh -c '
        git config -f /var/gerrit/etc/gerrit.config gerrit.canonicalWebUrl
"http://localhost:8080"
        git config -f /var/gerrit/etc/gerrit.config auth.type "OAUTH"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.client-id "YOUR_GITLAB_COM_APP_ID"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.client-secret "YOUR_GITLAB_COM_APP_SECRET"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.domain "gitlab.com"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.use-preferred-username "true"
        /var/gerrit/bin/gerrit.sh run
      '
```

```
volumes:
  gerrit_data:
EOF

# Start Gerrit
docker-compose up -d
```

Step 3: Test GitLab.com Integration

```
# Clone your GitLab.com repository
git clone https://gitlab.com/your-username/your-project.git
cd your-project

# Add Gerrit remote for testing
git remote add gerrit ssh://your-gitlab-username@localhost:29418/your-
username/your-project

# Install commit-msg hook
curl -Lo .git/hooks/commit-msg http://localhost:8080/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg

# Test workflow
echo "Local testing with GitLab.com" > test-file.txt
git add test-file.txt
git commit -m "Test local Gerrit with GitLab.com

Testing the integration between local Gerrit and GitLab.com.
"

# Submit for review
git push gerrit HEAD:refs/for/main

# Login to Gerrit: http://localhost:8080 → Should authenticate via GitLab.com
```

Option 2C: Network GitLab Instance Integration

For GitLab running on your company network (e.g., http://gitlab.company.local):

Step 1: Configure Network Access

```
# If using Docker Desktop on Windows/Mac, ensure network connectivity
# Add to docker-compose.yml if needed:
version: '3'
services:
  gerrit:
    image: gerritcodereview/gerrit:latest
    ports:
```

```
      - "8080:8080"
      - "29418:29418"
    extra_hosts:
      - "gitlab.company.local:192.168.1.100"  # Replace with actual IP
    # ... rest of configuration
```

Step 2: Configure for Network GitLab

```
# Update OAuth configuration for network GitLab
cat > docker-compose.yml << 'EOF'
version: '3'
services:
  gerrit:
    image: gerritcodereview/gerrit:latest
    ports:
      - "8080:8080"
      - "29418:29418"
    volumes:
      - gerrit_data:/var/gerrit
      - ./plugins:/var/gerrit/plugins
    extra_hosts:
      - "gitlab.company.local:192.168.1.100"  # Adjust IP as needed
    environment:
      - CANONICAL_WEB_URL=http://localhost:8080
    command: |
      sh -c '
        git config -f /var/gerrit/etc/gerrit.config gerrit.canonicalWebUrl
"http://localhost:8080"
        git config -f /var/gerrit/etc/gerrit.config auth.type "OAUTH"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.client-id "YOUR_NETWORK_GITLAB_APP_ID"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.client-secret "YOUR_NETWORK_GITLAB_APP_SECRET"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.domain "gitlab.company.local"
        git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-
provider-gitlab-oauth.use-preferred-username "true"
        /var/gerrit/bin/gerrit.sh run
      '

volumes:
  gerrit_data:
EOF
```

Step 3: Test Network GitLab Integration

```
# Test connectivity
curl http://gitlab.company.local
```

```
# Clone repository from network GitLab
git clone http://gitlab.company.local/your-group/your-project.git
cd your-project

# Add Gerrit remote
git remote add gerrit ssh://your-username@localhost:29418/your-group/your-
project

# Install hooks and test
curl -Lo .git/hooks/commit-msg http://localhost:8080/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg

# Make test change
echo "Network GitLab integration test" > network-test.txt
git add network-test.txt
git commit -m "Test network GitLab integration"
git push gerrit HEAD:refs/for/main
```

Advanced Testing Configurations

Multiple GitLab Instances

```
# You can configure multiple OAuth providers for testing different GitLab
instances
# Create separate configurations in gerrit.config:

git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-provider-
gitlab-oauth-local.client-id "LOCAL_APP_ID"
git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-provider-
gitlab-oauth-local.domain "gitlab.local"

git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-provider-
gitlab-oauth-com.client-id "GITLAB_COM_APP_ID"
git config -f /var/gerrit/etc/gerrit.config plugin.gerrit-oauth-provider-
gitlab-oauth-com.domain "gitlab.com"
```

Testing Replication

```
# Set up replication for testing (optional)
# Create replication.config inside container:
docker exec -it gerrit-docker_gerrit_1 sh -c '
cat > /var/gerrit/etc/replication.config << EOF
[remote "test-gitlab"]
    url = git@gitlab.com:\${name}.git
    push = +refs/heads/*:refs/heads/*
    authGroup = Administrators
```

```
EOF
'

# Restart Gerrit to load replication config
docker-compose restart
```

Troubleshooting Local Integrations

```
# Check Gerrit logs
docker-compose logs -f gerrit

# Check OAuth configuration
docker exec gerrit-docker_gerrit_1 cat /var/gerrit/etc/gerrit.config

# Test OAuth endpoint
curl "http://localhost:8080/login"

# Verify plugin installation
docker exec gerrit-docker_gerrit_1 ls -la /var/gerrit/plugins/

# Reset configuration if needed
docker-compose down
docker volume rm gerrit-docker_gerrit_data
# Then restart with fresh configuration
```

Option 3: Gerrit Server with GitLab Integration (Production Setup)

If you want to set up your own Gerrit server that integrates with your existing GitLab instance, this option provides a complete production-ready setup.

Prerequisites

- Linux server (Ubuntu 20.04+ recommended)
- Java 11 or 17 installed
- Access to your GitLab instance (gitlab.mycompany.com)
- Administrative privileges on the server

Step 1: Prepare the Environment

```
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install Java 11
sudo apt install openjdk-11-jdk -y

# Verify Java installation
```

```
java -version

# Create Gerrit user
sudo adduser gerrit --system --shell /bin/bash --gecos 'Gerrit Code Review' --
group --disabled-password --home /home/gerrit

# Create Gerrit installation directory
sudo mkdir -p /opt/gerrit
sudo chown gerrit:gerrit /opt/gerrit
```

Step 2: Download and Install Gerrit

```
# Switch to gerrit user
sudo su - gerrit

# Download Gerrit (replace with latest version)
cd /opt/gerrit
wget https://gerrit-releases.storage.googleapis.com/gerrit-3.8.0.war

# Create Gerrit site
java -jar gerrit-3.8.0.war init -d /opt/gerrit/review_site
```

Step 3: Configure Gerrit for GitLab Integration

During the init process, configure the following settings:

```
# Database settings (use H2 for simplicity or PostgreSQL/MySQL for production)
Database server type: h2

# Authentication settings
Authentication method: HTTP
Get username from custom HTTP header: REMOTE_USER
SSO logout URL: https://gitlab.mycompany.com/users/sign_out

# HTTP settings
Listen on address: *
Listen on port: 8080
Canonical URL: https://gerrit.mycompany.com/

# SSH settings
Listen on port: 29418
```

Step 4: Configure GitLab Authentication

Option A: LDAP Integration (Recommended for Enterprise)

Edit `/opt/gerrit/review_site/etc/gerrit.config`:

```
[gerrit]
    basePath = git
    canonicalWebUrl = https://gerrit.mycompany.com/
    serverId = gerrit-prod

[database]
    type = h2
    database = /opt/gerrit/review_site/db/ReviewDB

[auth]
    type = LDAP
    gitBasicAuthPolicy = LDAP

[ldap]
    server = ldap://ldap.mycompany.com
    username = cn=gerrit,ou=services,dc=mycompany,dc=com
    password = your-ldap-password
    accountBase = ou=users,dc=mycompany,dc=com
    accountPattern = (&(objectClass=person)(uid=${username}))
    accountFullName = displayName
    accountEmailAddress = mail
    groupBase = ou=groups,dc=mycompany,dc=com

[httpd]
    listenUrl = http://*:8080/

[sshd]
    listenAddress = *:29418

[cache]
    directory = cache

[container]
    user = gerrit
    javaHome = /usr/lib/jvm/java-11-openjdk-amd64
```

Option B: OAuth with GitLab

For OAuth integration, install the GitLab OAuth plugin:

```
# Download GitLab OAuth plugin
cd /opt/gerrit/review_site/plugins
wget https://github.com/davido/gerrit-oauth-
provider/releases/download/v3.8.0/gerrit-oauth-provider.jar

# Configure OAuth in gerrit.config
```

Add to `/opt/gerrit/review_site/etc/gerrit.config`:

```
[auth]
    type = OAUTH
    gitBasicAuthPolicy = OAUTH

[plugin "gerrit-oauth-provider-gitlab-oauth"]
    client-id = your-gitlab-app-id
    client-secret = your-gitlab-app-secret
    link-to-existing-openid-accounts = true
    domain = gitlab.mycompany.com
    use-preferred-username = true
```

Step 5: Set Up GitLab Application (for OAuth)

1. **In GitLab (https://gitlab.mycompany.com):**
    - Go to Admin Area → Applications
    - Create new application:
        - Name: "Gerrit Code Review"
        - Redirect URI: `https://gerrit.mycompany.com/oauth`
        - Scopes: `read_user`, `read_api`, `read_repository`
    - Note the Application ID and Secret
2. **Update Gerrit config with the credentials obtained above**

Step 6: Configure Reverse Proxy (Nginx)

Install and configure Nginx as reverse proxy:

```
# Install Nginx
sudo apt install nginx -y

# Create Gerrit site configuration
sudo tee /etc/nginx/sites-available/gerrit << 'EOF'
server {
    listen 80;
    server_name gerrit.mycompany.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name gerrit.mycompany.com;

    ssl_certificate /etc/ssl/certs/gerrit.mycompany.com.crt;
    ssl_certificate_key /etc/ssl/private/gerrit.mycompany.com.key;
```

```
    # SSL configuration
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-
RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;

    client_max_body_size 50M;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # For GitLab authentication
        proxy_set_header REMOTE_USER $remote_user;
    }
}
EOF

# Enable the site
sudo ln -s /etc/nginx/sites-available/gerrit /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

Step 7: Set Up SSL Certificates

```
# Option A: Let's Encrypt (recommended)
sudo apt install certbot python3-certbot-nginx -y
sudo certbot --nginx -d gerrit.mycompany.com

# Option B: Use your company's SSL certificates
# Copy your certificates to:
# /etc/ssl/certs/gerrit.mycompany.com.crt
# /etc/ssl/private/gerrit.mycompany.com.key
```

Step 8: Create Systemd Service

```
# Create systemd service file
sudo tee /etc/systemd/system/gerrit.service << 'EOF'
[Unit]
Description=Gerrit Code Review
After=network.target

[Service]
Type=notify
```

```
User=gerrit
Group=gerrit
ExecStart=/usr/bin/java -jar /opt/gerrit/gerrit-3.8.0.war daemon -d
/opt/gerrit/review_site
ExecReload=/bin/kill -HUP $MAINPID
KillMode=mixed
SuccessExitStatus=3
RestartForceExitStatus=3

[Install]
WantedBy=multi-user.target
EOF

# Reload systemd and start Gerrit
sudo systemctl daemon-reload
sudo systemctl enable gerrit
sudo systemctl start gerrit

# Check status
sudo systemctl status gerrit
```

Step 9: Configure GitLab Integration for Project Mirroring

In GitLab:

1. **Set up push mirroring:**
   - Go to your GitLab project
   - Settings → Repository → Mirroring repositories
   - Add mirror: `ssh://gerrit@gerrit.mycompany.com:29418/your-project.git`
   - Mirror direction: Push
   - Authentication: SSH public key
2. **Add Gerrit's SSH key to GitLab:**

```
# Generate SSH key for Gerrit
sudo su - gerrit
ssh-keygen -t rsa -b 4096 -C "gerrit@mycompany.com"
cat ~/.ssh/id_rsa.pub
```

   - Add this public key to GitLab project's deploy keys

In Gerrit:

1. **Create project matching GitLab repository:**
   - Access Gerrit UI: https://gerrit.mycompany.com
   - Browse → Repositories → Create New
   - Project name should match GitLab project path

2. **Set up replication back to GitLab:**

   Create `/opt/gerrit/review_site/etc/replication.config`:

```
[remote "gitlab"]
    url = git@gitlab.mycompany.com:${name}.git
    push = +refs/heads/*:refs/heads/*
    push = +refs/tags/*:refs/tags/*
    authGroup = Administrators
    createMissingRepositories = false
```

Step 10: Configure Project Access and Permissions

```
# Access Gerrit as admin
# Go to Browse → Repositories → All-Projects → Access

# Set up basic permissions:
# - Reference: refs/heads/*
#   - Read: Registered Users
#   - Label Code-Review: Registered Users (-2 to +2)
#   - Label Verified: Continuous Integration Tools (-1 to +1)
#   - Submit: Project Owners
#   - Push: Registered Users
#   - Push Merge Commit: Project Owners

# Reference: refs/for/refs/heads/*
#   - Push: Registered Users
```

Step 11: Test the Integration

```
# Clone from GitLab
git clone https://gitlab.mycompany.com/your-group/your-project.git
cd your-project

# Add Gerrit remote
git remote add gerrit ssh://your-username@gerrit.mycompany.com:29418/your-
project

# Install commit-msg hook
curl -Lo .git/hooks/commit-msg https://gerrit.mycompany.com/tools/hooks/commit-
msg
chmod +x .git/hooks/commit-msg

# Make a test change
echo "Test change" > test.txt
git add test.txt
```

```
git commit -m "Test Gerrit integration

This change tests the integration between GitLab and Gerrit.
"

# Submit to Gerrit for review
git push gerrit HEAD:refs/for/main

# After review and merge, changes should replicate back to GitLab
```

Option 4: Gerrit Server with Public GitLab Integration (GitLab.com)

If you want to set up your own Gerrit server that integrates with public GitLab (https://gitlab.com/), this option provides integration with the hosted GitLab service.

Prerequisites

- Linux server (Ubuntu 20.04+ recommended)
- Java 11 or 17 installed
- GitLab.com account with projects
- Domain name for your Gerrit server (e.g., gerrit.yourcompany.com)
- Administrative privileges on the server

Step 1: Prepare the Environment

```
# Update system packages
sudo apt update && sudo apt upgrade -y

# Install Java 11
sudo apt install openjdk-11-jdk -y

# Verify Java installation
java -version

# Create Gerrit user
sudo adduser gerrit --system --shell /bin/bash --gecos 'Gerrit Code Review' --
group --disabled-password --home /home/gerrit

# Create Gerrit installation directory
sudo mkdir -p /opt/gerrit
sudo chown gerrit:gerrit /opt/gerrit
```

Step 2: Download and Install Gerrit

```
# Switch to gerrit user
sudo su - gerrit
```

```
# Download Gerrit (replace with latest version)
cd /opt/gerrit
wget https://gerrit-releases.storage.googleapis.com/gerrit-3.8.0.war

# Create Gerrit site
java -jar gerrit-3.8.0.war init -d /opt/gerrit/review_site
```

Step 3: Configure Gerrit for GitLab.com Integration

During the init process, configure the following settings:

```
# Database settings (use H2 for simplicity or PostgreSQL/MySQL for production)
Database server type: h2

# Authentication settings
Authentication method: OAUTH
Get username from custom HTTP header: [leave empty]
SSO logout URL: https://gitlab.com/users/sign_out

# HTTP settings
Listen on address: *
Listen on port: 8080
Canonical URL: https://gerrit.yourcompany.com/

# SSH settings
Listen on port: 29418
```

Step 4: Set Up GitLab.com OAuth Application

1. **Create OAuth Application in GitLab.com:**
   - Go to https://gitlab.com/-/profile/applications
   - Click "New application"
   - Application details:
     - Name: "Gerrit Code Review"
     - Redirect URI: `https://gerrit.yourcompany.com/oauth`
     - Confidential: ☑ (checked)
     - Scopes: Select `read_user`, `read_api`, `read_repository`
   - Click "Save application"
   - **Note down the Application ID and Secret**

Step 5: Configure GitLab.com OAuth in Gerrit

Install the GitLab OAuth plugin and configure:

```
# Download GitLab OAuth plugin
cd /opt/gerrit/review_site/plugins
wget https://github.com/davido/gerrit-oauth-
provider/releases/download/v3.8.0/gerrit-oauth-provider.jar
```

Edit `/opt/gerrit/review_site/etc/gerrit.config`:

```
[gerrit]
    basePath = git
    canonicalWebUrl = https://gerrit.yourcompany.com/
    serverId = gerrit-gitlab-integration

[database]
    type = h2
    database = /opt/gerrit/review_site/db/ReviewDB

[auth]
    type = OAUTH
    gitBasicAuthPolicy = OAUTH

[plugin "gerrit-oauth-provider-gitlab-oauth"]
    client-id = your-gitlab-app-id-from-step-4
    client-secret = your-gitlab-app-secret-from-step-4
    link-to-existing-openid-accounts = true
    domain = gitlab.com
    use-preferred-username = true
    fix-legacy-user-id = true

[httpd]
    listenUrl = http://*:8080/

[sshd]
    listenAddress = *:29418

[cache]
    directory = cache

[container]
    user = gerrit
    javaHome = /usr/lib/jvm/java-11-openjdk-amd64
```

Step 6: Configure Reverse Proxy (Nginx)

```
# Install Nginx
sudo apt install nginx -y

# Create Gerrit site configuration
```

```
sudo tee /etc/nginx/sites-available/gerrit << 'EOF'
server {
    listen 80;
    server_name gerrit.yourcompany.com;
    return 301 https://$server_name$request_uri;
}

server {
    listen 443 ssl http2;
    server_name gerrit.yourcompany.com;

    ssl_certificate /etc/ssl/certs/gerrit.yourcompany.com.crt;
    ssl_certificate_key /etc/ssl/private/gerrit.yourcompany.com.key;

    # SSL configuration
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-
RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;

    client_max_body_size 50M;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Server $host;
    }
}
EOF

# Enable the site
sudo ln -s /etc/nginx/sites-available/gerrit /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

Step 7: Set Up SSL Certificates

```
# Option A: Let's Encrypt (recommended for public domains)
sudo apt install certbot python3-certbot-nginx -y
sudo certbot --nginx -d gerrit.yourcompany.com

# Option B: Use your own SSL certificates
# Copy your certificates to:
# /etc/ssl/certs/gerrit.yourcompany.com.crt
# /etc/ssl/private/gerrit.yourcompany.com.key
```

Step 8: Create Systemd Service

```
# Create systemd service file
sudo tee /etc/systemd/system/gerrit.service << 'EOF'
[Unit]
Description=Gerrit Code Review
After=network.target

[Service]
Type=notify
User=gerrit
Group=gerrit
ExecStart=/usr/bin/java -jar /opt/gerrit/gerrit-3.8.0.war daemon -d
/opt/gerrit/review_site
ExecReload=/bin/kill -HUP $MAINPID
KillMode=mixed
SuccessExitStatus=3
RestartForceExitStatus=3

[Install]
WantedBy=multi-user.target
EOF

# Reload systemd and start Gerrit
sudo systemctl daemon-reload
sudo systemctl enable gerrit
sudo systemctl start gerrit

# Check status
sudo systemctl status gerrit
```

Step 9: Configure GitLab.com Integration for Project Mirroring

In GitLab.com:

1. **Set up push mirroring (GitLab Premium feature):**
    - Go to your GitLab.com project
    - Settings → Repository → Mirroring repositories
    - Add mirror: `ssh://gerrit@gerrit.yourcompany.com:29418/your-project.git`
    - Mirror direction: Push
    - Authentication: SSH public key
2. **Alternative: Webhook for push notifications (Free tier):**
    - Go to Settings → Webhooks
    - URL: `https://gerrit.yourcompany.com/hooks/gitlab`
    - Trigger: Push events, Merge request events
    - Secret Token: (optional, for security)

3. **Add Gerrit's SSH key to GitLab.com:**

```
# Generate SSH key for Gerrit
sudo su - gerrit
ssh-keygen -t rsa -b 4096 -C "gerrit@yourcompany.com"
cat ~/.ssh/id_rsa.pub
```

   - In GitLab.com: Go to your project → Settings → Deploy keys
   - Add the public key with write access

In Gerrit:

1. **Create project matching GitLab repository:**

   - Access Gerrit UI: https://gerrit.yourcompany.com
   - Browse → Repositories → Create New
   - Project name should match GitLab project path (e.g., `username/project-name`)

2. **Set up replication back to GitLab.com:**

   Create `/opt/gerrit/review_site/etc/replication.config`:

```
[remote "gitlab"]
    url = git@gitlab.com:${name}.git
    push = +refs/heads/*:refs/heads/*
    push = +refs/tags/*:refs/tags/*
    authGroup = Administrators
    createMissingRepositories = false
    replicationDelay = 0
    timeout = 30
```

Step 10: Configure Access Tokens for GitLab.com API

For advanced integration features, create a GitLab.com personal access token:

1. **In GitLab.com:**

   - Go to User Settings → Access Tokens
   - Create token with scopes: `api`, `read_repository`, `write_repository`
   - Note the token

2. **Configure in Gerrit (optional):**

```
# Add to secure.config for API access
sudo tee -a /opt/gerrit/review_site/etc/secure.config << 'EOF'
[gitlab]
```

```
    token = your-gitlab-personal-access-token
EOF
```

Step 11: Test the Integration

```
# Test OAuth login
# 1. Go to https://gerrit.yourcompany.com
# 2. Click "Sign In" → Should redirect to GitLab.com
# 3. Authorize the application
# 4. Should redirect back to Gerrit with your GitLab.com identity

# Test repository integration
# Clone from GitLab.com
git clone https://gitlab.com/your-username/your-project.git
cd your-project

# Add Gerrit remote
git remote add gerrit ssh://your-gitlab-
username@gerrit.yourcompany.com:29418/your-username/your-project

# Install commit-msg hook
curl -Lo .git/hooks/commit-msg
https://gerrit.yourcompany.com/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg

# Make a test change
echo "Test GitLab.com integration" > gitlab-test.txt
git add gitlab-test.txt
git commit -m "Test GitLab.com integration

This change tests the integration between Gerrit and GitLab.com.
"

# Submit to Gerrit for review
git push gerrit HEAD:refs/for/main

# After review and merge, changes should replicate back to GitLab.com
```

Step 12: Additional Configuration for GitLab.com

Configure project permissions for public GitLab:

```
# In Gerrit UI: Browse → Repositories → All-Projects → Access

# Add GitLab.com users group permissions:
# - Reference: refs/heads/*
#   - Read: Registered Users
```

```
#   - Label Code-Review: Registered Users (-2 to +2)
#   - Submit: Project Owners
#   - Push: Registered Users (for direct pushes)

# Reference: refs/for/refs/heads/*
#   - Push: Registered Users (for reviews)

# Reference: refs/tags/*
#   - Read: Registered Users
#   - Create: Project Owners
```

Set up automatic project creation (optional):

Edit `/opt/gerrit/review_site/etc/gerrit.config`:

```
[receive]
    enableSignedPush = false

[gerrit]
    createChangeForAllNotInTarget = true

[repository "*"]
    defaultSubmitType = MERGE_IF_NECESSARY
```

Verification Steps

Test 1: Basic Gerrit Workflow

```
# Make a change
echo "Hello Gerrit!" > hello.txt
git add hello.txt
git commit -m "Add hello file

This is a test change for Gerrit setup verification.
"

# Submit to Gerrit
git push origin HEAD:refs/for/main

# Check Gerrit UI - should see your change
```

Test 2: GitLab Integration (for Option 3)

```
# Verify authentication works
curl -u username:password https://gerrit.mycompany.com/a/accounts/self
```

```
# Test change replication to GitLab
# Submit a change to Gerrit, merge it, then check GitLab repository

# Verify SSH access
ssh -p 29418 gerrit.mycompany.com gerrit version
```

Test 3: End-to-End Workflow

```
# Start from GitLab repository
git clone https://gitlab.mycompany.com/your-group/your-project.git
cd your-project

# Add Gerrit remote and hooks
git remote add gerrit ssh://username@gerrit.mycompany.com:29418/your-project
curl -Lo .git/hooks/commit-msg https://gerrit.mycompany.com/tools/hooks/commit-
msg
chmod +x .git/hooks/commit-msg

# Make change and submit for review
echo "Integration test" > integration-test.txt
git add integration-test.txt
git commit -m "Add integration test file"
git push gerrit HEAD:refs/for/main

# Review in Gerrit UI, merge, then verify in GitLab
```

Test 4: GitLab.com Integration (for Option 4)

```
# Test OAuth authentication with GitLab.com
# 1. Open https://gerrit.yourcompany.com
# 2. Click "Sign In" - should redirect to GitLab.com
# 3. Authorize application - should redirect back with your GitLab.com profile

# Test API connectivity
curl -H "Authorization: Bearer YOUR_GITLAB_TOKEN"
https://gitlab.com/api/v4/user

# Test SSH connectivity to GitLab.com
ssh -T git@gitlab.com

# Test replication
# Submit change → Review → Merge → Check GitLab.com repository

# Verify webhook (if configured)
curl -X POST https://gerrit.yourcompany.com/hooks/gitlab \
```

```
  -H "Content-Type: application/json" \
  -d '{"test": "webhook"}'
```

Test 5: End-to-End Public GitLab Workflow

```
# Start from GitLab.com repository
git clone https://gitlab.com/your-username/your-project.git
cd your-project

# Add Gerrit remote and hooks
git remote add gerrit ssh://your-gitlab-
username@gerrit.yourcompany.com:29418/your-username/your-project
curl -Lo .git/hooks/commit-msg
https://gerrit.yourcompany.com/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg

# Make change and submit for review
echo "Public GitLab integration test" > public-gitlab-test.txt
git add public-gitlab-test.txt
git commit -m "Add public GitLab integration test"
git push gerrit HEAD:refs/for/main

# Login to Gerrit with GitLab.com OAuth
# Review the change in Gerrit UI
# Merge and verify replication back to GitLab.com
```

---

2. Change-Based Workflow

The Big Picture

Unlike GitHub's branch-based workflow, Gerrit works with **individual changes** (commits).

```
GitHub Workflow:
feature-branch → Pull Request → Merge entire branch

Gerrit Workflow:
single-commit → Change Request → Review → Merge single change
```

Real-World Example

Scenario: You need to fix a bug in a login function.

Traditional Git:

```
git checkout -b fix-login-bug
# make changes
git commit -m "Fix login validation"
git push origin fix-login-bug
# Create PR on GitHub
```

Gerrit Way:

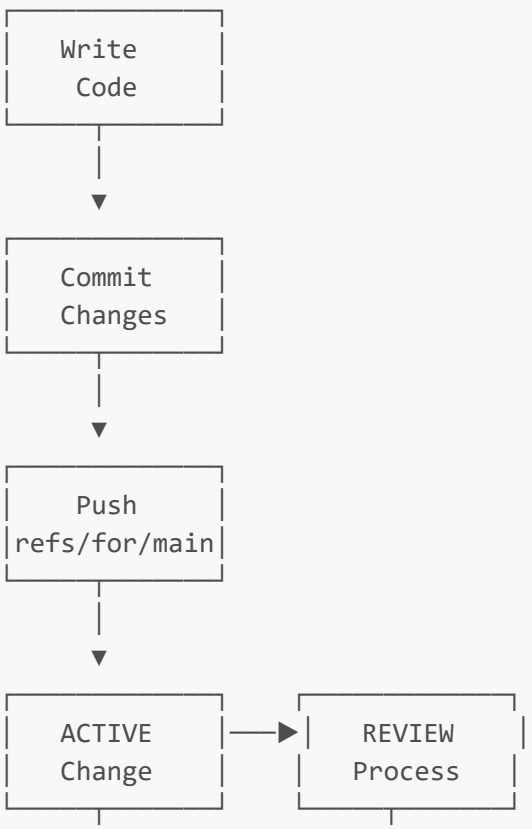
```
# Work directly on main branch (or any branch)
git checkout main
# make changes
git commit -m "Fix login validation"
git push origin HEAD:refs/for/main
# Gerrit automatically creates a change for review
```

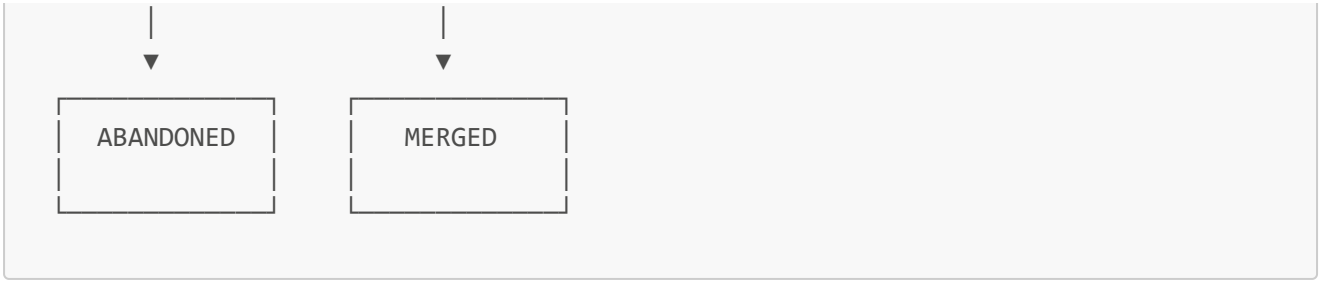
Key Insight

- **One commit = One change = One review**
- No branches needed for simple changes
- Each change gets a unique Change-Id

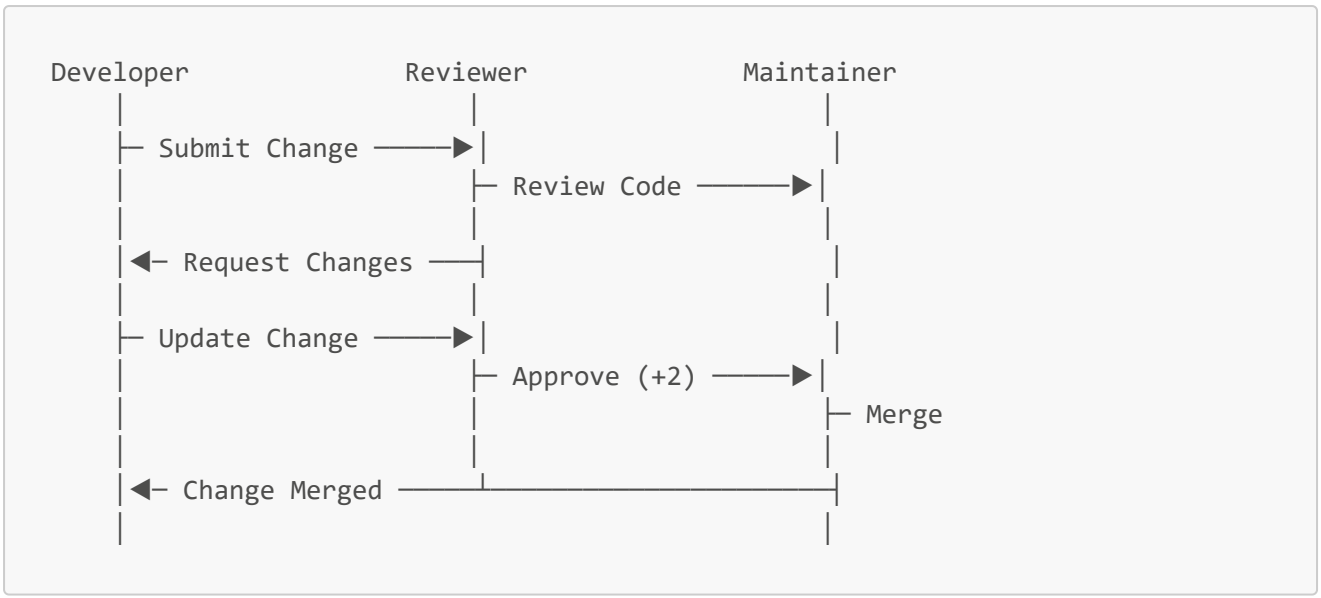
Visual Workflow Diagrams

Change Lifecycle Flowchart

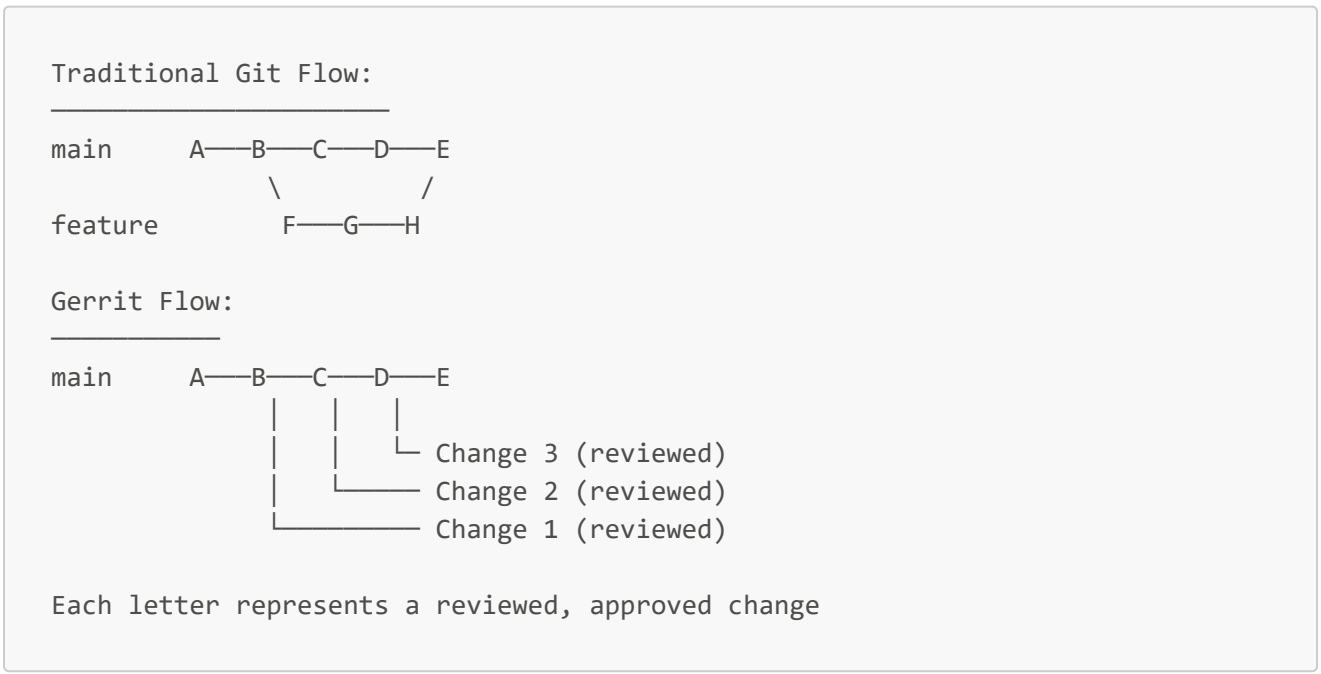




Review Process Flow



Git vs Gerrit Comparison



---

3. The Review Process

The Review Lifecycle

```
1. DRAFT (optional) → 2. ACTIVE → 3. REVIEWED → 4. MERGED
                              ↓
                         5. ABANDONED
```

Review Scores Explained

Score	Meaning	Can Merge?
+2	Looks good to me, approved	☑ Yes
+1	Looks good to me, but someone else must approve	❌ No
0	No score	❌ No
-1	I would prefer this is not merged	❌ No
-2	This shall not be merged	❌ No

Real-World Review Example

Bad Review Comment:

```
"This code is wrong."
```

Good Review Comment:

```
"Consider using a switch statement instead of multiple if-else
statements for better readability. Also, this might throw a
NullPointerException if user is null on line 23."
```

---

4. Essential Commands

The 5 Commands You'll Use 90% of the Time

1. Submit a Change for Review

```
git push origin HEAD:refs/for/main
```

When to use: Every time you want to submit code for review

2. Update an Existing Change

```
git commit --amend
git push origin HEAD:refs/for/main
```

When to use: After receiving review feedback

3. Download a Change for Review

```
git fetch origin refs/changes/12/34/5 && git checkout FETCH_HEAD
```

When to use: When you want to test someone else's change

4. Check Status

```
git log --oneline -5
```

When to use: To see recent commits and their Change-Ids

5. Rebase Before Submit

```
git rebase origin/main
git push origin HEAD:refs/for/main
```

When to use: When your change conflicts with newer changes

---

5. Change States

Understanding Change States

```
NEW → ACTIVE → MERGED
  ↓      ↓        ↑
DRAFT    ↓        ↑
       ABANDONED
```

What Each State Means

- **NEW/DRAFT:** Change exists but not ready for review
- **ACTIVE:** Change is ready and under review
- **MERGED:** Change has been integrated into the target branch
- **ABANDONED:** Change has been discarded

Real-World State Transitions

Scenario: You submit a change with a typo in the commit message.

1. **Submit change** → State: ACTIVE
2. **Realize typo** → Amend commit message
3. **Push updated change** → State: Still ACTIVE (same change, new version)
4. **Get +2 review** → State: Ready to merge
5. **Merge change** → State: MERGED

---

6. Merge Strategies

The Three Ways Changes Get Merged

1. Fast Forward (Preferred)

```
Before: A---B---C (main)
After:  A---B---C---D (main, your change)
```

When it happens: Your change is based on the latest main branch

2. Merge Commit

```
Before: A---B---C (main)
        \
         D (your change)
After:  A---B---C---E (main)
         \         /
          D-------/
```

When it happens: Your change conflicts with newer changes

3. Cherry Pick (Common)

```
Before: A---B---C---X (main, new changes)
        \
         D (your change, based on B)
After:  A---B---C---X---D' (main, your change rebased)
```

When it happens: Gerrit rebases your change automatically

---

7. Common Scenarios

Scenario 1: "My Change Won't Merge"

Problem: You see "Cannot merge due to conflicts"

Solution:

```
# 1. Fetch latest changes
git fetch origin

# 2. Rebase your change
git rebase origin/main

# 3. Resolve conflicts (if any)
git add .
git rebase --continue

# 4. Push updated change
git push origin HEAD:refs/for/main
```

Scenario 2: "I Need to Update My Change"

Problem: Reviewer asked for changes

Solution:

```
# 1. Make the requested changes
# edit files...

# 2. Amend the commit (don't create new commit!)
git add .
git commit --amend

# 3. Push updated change
git push origin HEAD:refs/for/main
```

Scenario 3: "I Want to Review Someone's Change"

Problem: How do I test someone else's code?

Solution:

```
# 1. Find the change number (e.g., 12345) in Gerrit UI
# 2. Download the change
git fetch origin refs/changes/45/12345/3 && git checkout FETCH_HEAD

# 3. Test the change
# run tests, try the feature...
```

```
# 4. Go back to your branch
git checkout main
```

Additional Real-World Examples

Example 1: Your First Code Review Submission

Context: You're a new developer who needs to fix a typo in documentation.

Step-by-Step:

1. **Setup your environment:**

```
# Clone the repository
git clone https://gerrit.example.com/myproject
cd myproject

# Install commit-msg hook
curl -Lo .git/hooks/commit-msg
https://gerrit.example.com/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg
```

2. **Make your change:**

```
# Edit the file
notepad README.md  # Fix typo: "recieve" → "receive"
```

3. **Commit and submit:**

```
git add README.md
git commit -m "Fix typo in README.md

Changed 'recieve' to 'receive' in installation instructions.
"

git push origin HEAD:refs/for/main
```

4. **What happens next:**
   - Gerrit creates a change (e.g., Change 12345)
   - You get a URL to the change in Gerrit UI
   - Reviewers can now see and review your change

Example 2: Handling Review Feedback

Context: A reviewer found issues with your code and left comments.

Review Comments:

```
Line 23: Consider using Optional<String> instead of null checks
Line 45: This method is too long, consider breaking it down
Line 67: Add JavaDoc comment for this public method
```

Step-by-Step Response:

1. **Address the feedback:**

   ```
   # Make the requested changes
   # Edit UserService.java:
   # - Line 23: Refactor to use Optional<String>
   # - Line 45: Split method into smaller methods
   # - Line 67: Add JavaDoc comment
   ```

2. **Update your change:**

   ```
   git add UserService.java
   git commit --amend  # This updates the existing commit
   git push origin HEAD:refs/for/main
   ```

3. **Respond to reviewer:**
   In Gerrit UI, add a comment:

   ```
   "Thanks for the review! I've addressed all the points:
   - Refactored to use Optional<String> for null safety
   - Split the large method into validateUser() and processUser()
   - Added comprehensive JavaDoc documentation

   Please take another look when you have time."
   ```

Example 3: Working with Dependent Changes

Context: You need to make two changes where the second depends on the first.

Changes:

1. Add new utility method

2. Use the utility method in business logic

Step-by-Step:

1. **Submit first change:**

```
# Make first change
git add Utils.java
git commit -m "Add validation utility method

Add isValidEmail() method to Utils class for email validation.
This will be used by user registration and profile update features.
"

git push origin HEAD:refs/for/main
# This creates Change 34567
```

2. **Create second change (depends on first):**

```
# Don't reset! Keep building on the same branch
# Make second change
git add UserService.java
git commit -m "Use Utils.isValidEmail() in user registration

Replace custom email validation with centralized utility method.
This ensures consistent validation across the application.

Depends-On: I1234567890abcdef (Change-Id of first change)
"

git push origin HEAD:refs/for/main
# This creates Change 34568 that depends on 34567
```

3. **What happens:**

   - First change gets reviewed and merged
   - Second change automatically rebases and becomes ready for review
   - Gerrit shows the dependency relationship in the UI

```
---

## 8. Troubleshooting

### Problem: "Missing Change-Id"

**Error message:** `missing Change-Id in commit message footer`
```

```
**Solution:**
```bash
# Install the commit-msg hook
curl -Lo .git/hooks/commit-msg http://your-gerrit-server/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg

# Amend your commit to add Change-Id
git commit --amend
```

Problem: "Push Rejected"

Error message: `[remote rejected] HEAD -> refs/for/main`

Common causes & solutions:

1. **No permissions:** Contact your Gerrit admin
2. **Branch doesn't exist:** Check if you're pushing to the right branch
3. **Commit message issues:** Ensure proper format and Change-Id

Advanced Troubleshooting Guide

Issue: "Cannot connect to Gerrit"

Solutions:

```bash
# Check if Gerrit is running
curl http://localhost:8080

# Check firewall settings
sudo ufw status
sudo ufw allow 8080
sudo ufw allow 29418

# Check Gerrit logs
tail -f ~/gerrit_site/logs/error_log
```

Issue: "SSH key not working"

Solutions:

```bash
# Generate new SSH key
ssh-keygen -t rsa -b 4096 -C "your.email@example.com"

# Add to ssh-agent
eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_rsa
```
```

```
# Add public key to Gerrit
# In Gerrit UI: Settings → SSH Keys → Add Key
cat ~/.ssh/id_rsa.pub

# Test SSH connection
ssh -p 29418 localhost
```

Issue: "Commit-msg hook not working"

Solutions:

```
# Check hook exists and is executable
ls -la .git/hooks/commit-msg

# Re-download hook
curl -Lo .git/hooks/commit-msg http://localhost:8080/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg

# Test manually
echo "Test message" > /tmp/test-msg
.git/hooks/commit-msg /tmp/test-msg
cat /tmp/test-msg  # Should have Change-Id
```

Issue: "Change already exists"

Solution:

```
# You're trying to push the same commit again
# Either abandon the existing change or amend your commit
git commit --amend
git push origin HEAD:refs/for/main
```

Issue: "Permission denied"

Common causes:

1. **No push permissions:** Contact admin
2. **Wrong branch name:** Check if branch exists
3. **Authentication failed:** Verify credentials

Issue: "Merge conflicts during rebase"

Solution:

```
git fetch origin
git rebase origin/main
# Resolve conflicts in editor
git add .
git rebase --continue
git push origin HEAD:refs/for/main
```

Diagnostic Commands

```
# Check Git configuration
git config --list

# Check remote configuration
git remote -v

# Check branch tracking
git branch -vv

# Check commit-msg hook
ls -la .git/hooks/commit-msg

# View Git log with graph
git log --oneline --graph -10

# Check reflog
git reflog
```

GitLab Integration Troubleshooting

Issue: "Authentication failed with GitLab"

Solutions:

```
# For LDAP authentication
# Check LDAP connectivity
ldapsearch -x -H ldap://ldap.mycompany.com -D
"cn=gerrit,ou=services,dc=mycompany,dc=com" -W -b
"ou=users,dc=mycompany,dc=com"

# Check Gerrit logs for LDAP errors
tail -f /opt/gerrit/review_site/logs/error_log | grep -i ldap

# For OAuth authentication
# Verify GitLab application settings
curl -H "Authorization: Bearer YOUR_GITLAB_TOKEN"
https://gitlab.mycompany.com/api/v4/applications
```

Issue: "Replication to GitLab not working"

Solutions:

```
# Check replication plugin status
ssh -p 29418 gerrit.mycompany.com gerrit plugin ls

# Check replication logs
tail -f /opt/gerrit/review_site/logs/replication_log

# Test SSH connectivity to GitLab
ssh -T git@gitlab.mycompany.com

# Force replication manually
ssh -p 29418 gerrit.mycompany.com gerrit plugin reload replication
ssh -p 29418 gerrit.mycompany.com gerrit replicate --all
```

Issue: "SSL certificate problems"

Solutions:

```
# Check certificate validity
openssl s_client -connect gerrit.mycompany.com:443 -servername
gerrit.mycompany.com

# Verify certificate chain
openssl verify -CAfile /etc/ssl/certs/ca-certificates.crt
/etc/ssl/certs/gerrit.mycompany.com.crt

# Check Nginx configuration
sudo nginx -t
sudo systemctl status nginx
```

Issue: "Project creation fails"

Solutions:

```
# Check Gerrit permissions
# In Gerrit UI: Browse → Repositories → All-Projects → Access
# Ensure "Create Repository" permission is granted to appropriate groups

# Check filesystem permissions
ls -la /opt/gerrit/review_site/git/
sudo chown -R gerrit:gerrit /opt/gerrit/review_site/
```

```
# Check Gerrit logs
tail -f /opt/gerrit/review_site/logs/error_log
```

Issue: "Changes not syncing between GitLab and Gerrit"

Solutions:

```
# Check GitLab push mirroring status
# In GitLab: Project → Settings → Repository → Mirroring repositories

# Verify SSH keys are properly configured
ssh-add -l  # List SSH keys
ssh -T git@gitlab.mycompany.com  # Test GitLab SSH

# Check Gerrit receive hooks
ls -la /opt/gerrit/review_site/hooks/

# Manual sync test
git push gerrit main:main  # Push directly to main branch
```

Common Configuration Issues

Gerrit config (`/opt/gerrit/review_site/etc/gerrit.config`):

```
# Ensure canonical URL matches your setup
[gerrit]
    canonicalWebUrl = https://gerrit.mycompany.com/

# For GitLab integration, auth type should be LDAP or OAUTH
[auth]
    type = LDAP  # or OAUTH

# HTTP settings for reverse proxy
[httpd]
    listenUrl = proxy-https://127.0.0.1:8080/
```

Replication config (`/opt/gerrit/review_site/etc/replication.config`):

```
[remote "gitlab"]
    url = git@gitlab.mycompany.com:${name}.git
    push = +refs/heads/*:refs/heads/*
    push = +refs/tags/*:refs/tags/*
    authGroup = Administrators
    replicationDelay = 0
    timeout = 30
```

GitLab.com Integration Troubleshooting

Issue: "OAuth authentication fails with GitLab.com"

Solutions:

```
# Check OAuth application configuration in GitLab.com
# Go to https://gitlab.com/-/profile/applications
# Verify:
# - Redirect URI matches: https://gerrit.yourcompany.com/oauth
# - Application is not revoked
# - Scopes include: read_user, read_api, read_repository

# Check Gerrit OAuth plugin configuration
grep -A 10 "gerrit-oauth-provider-gitlab-oauth"
/opt/gerrit/review_site/etc/gerrit.config

# Test OAuth endpoint manually
curl -v "https://gitlab.com/oauth/authorize?
client_id=YOUR_CLIENT_ID&redirect_uri=https://gerrit.yourcompany.com/oauth&resp
onse_type=code&scope=read_user"

# Check Gerrit logs for OAuth errors
tail -f /opt/gerrit/review_site/logs/error_log | grep -i oauth
```

Issue: "GitLab.com API rate limiting"

Solutions:

```
# Check rate limit status
curl -H "Authorization: Bearer YOUR_TOKEN" -I https://gitlab.com/api/v4/user

# Look for headers:
# RateLimit-Remaining: 1000
# RateLimit-Reset: 1609459200

# Configure rate limiting in replication.config
[remote "gitlab"]
    replicationDelay = 5
    timeout = 60

# Use personal access token for higher limits
# Premium/Ultimate plans have higher API limits
```

Issue: "SSH key authentication fails with GitLab.com"

Solutions:

```
# Test SSH connection
ssh -T git@gitlab.com
# Should return: "Welcome to GitLab, @username!"

# Check SSH key format (GitLab.com supports RSA, ECDSA, ED25519)
ssh-keygen -l -f ~/.ssh/id_rsa.pub

# Verify key is added to GitLab.com
curl -H "Authorization: Bearer YOUR_TOKEN" https://gitlab.com/api/v4/user/keys

# Test with verbose SSH
ssh -vT git@gitlab.com

# Check if key has proper permissions in GitLab.com
# Project → Settings → Deploy keys → Ensure "Write access allowed" is checked
```

Issue: "Repository mirroring not working (GitLab.com Free tier)"

Solutions:

```
# GitLab.com Free tier doesn't support push mirroring
# Use webhook-based integration instead

# Set up webhook in GitLab.com project
# Settings → Webhooks → Add webhook
# URL: https://gerrit.yourcompany.com/hooks/gitlab
# Triggers: Push events, Merge request events

# Alternative: Use GitLab CI/CD to push to Gerrit
# Create .gitlab-ci.yml:
cat > .gitlab-ci.yml << 'EOF'
push_to_gerrit:
  stage: deploy
  script:
    - git remote add gerrit ssh://user@gerrit.yourcompany.com:29418/project
    - git push gerrit HEAD:refs/for/main
  only:
    - main
EOF
```

Issue: "Project creation fails on GitLab.com"

Solutions:

```
# Check GitLab.com project visibility
# Private projects require proper access tokens

# Verify project path matches exactly
# GitLab.com: username/project-name
# Gerrit: username/project-name (case sensitive)

# Check namespace permissions
curl -H "Authorization: Bearer YOUR_TOKEN" \
  https://gitlab.com/api/v4/namespaces

# For organization projects, ensure token has appropriate permissions
```

Issue: "Webhook not receiving events"

Solutions:

```
# Check webhook delivery in GitLab.com
# Project → Settings → Webhooks → Click on webhook → View delivery attempts

# Verify webhook endpoint is accessible
curl -X POST https://gerrit.yourcompany.com/hooks/gitlab \
  -H "Content-Type: application/json" \
  -d '{"object_kind": "push", "test": true}'

# Check Gerrit logs for webhook events
tail -f /opt/gerrit/review_site/logs/httpd_log | grep webhook

# Common webhook issues:
# - SSL certificate validation
# - Firewall blocking GitLab.com IPs
# - Nginx configuration issues
```

Common GitLab.com Configuration Issues

OAuth Application Settings:

```
Application Name: Gerrit Code Review
Redirect URI: https://gerrit.yourcompany.com/oauth
Confidential: ☑ Yes
Scopes: read_user, read_api, read_repository
```

Gerrit OAuth Configuration:

```
```

```
[plugin "gerrit-oauth-provider-gitlab-oauth"]
    client-id = gitlab-app-id-from-oauth-app
    client-secret = gitlab-app-secret-from-oauth-app
    domain = gitlab.com
    use-preferred-username = true
    fix-legacy-user-id = true
```

GitLab.com Deploy Key Configuration:

- Key Type: SSH Key
- Title: Gerrit Integration
- Key: (Gerrit's public SSH key)
- Write access allowed: ☑ Yes

---

9. Practice Exercises

Exercise 1: Setup and First Change (Beginner)

Goal

Learn how to set up Gerrit and submit your first change.

Prerequisites

- Git installed
- Access to a Gerrit server (or use a local setup)

Steps

1. **Clone a repository:**

```
git clone https://gerrit.example.com/practice-repo
cd practice-repo
```

2. **Install commit-msg hook:**

```
curl -Lo .git/hooks/commit-msg
https://gerrit.example.com/tools/hooks/commit-msg
chmod +x .git/hooks/commit-msg
```

3. **Make a simple change:**

```
# Create or edit a file
echo "Hello Gerrit!" > hello.txt

# Stage and commit
git add hello.txt
git commit -m "Add hello.txt file

This is my first change submitted to Gerrit for review.
"
```

4. **Submit to Gerrit:**

```
git push origin HEAD:refs/for/main
```

Success Criteria

- ☐ Change appears in Gerrit UI
- ☐ Change has a Change-Id
- ☐ You can see the change details in browser

Exercise 2: Handle Review Feedback (Intermediate)

Goal

Learn how to update a change based on review feedback.

Setup

Use the change from Exercise 1 or create a new one.

Steps

1. **Create a change that needs improvement:**

```
# Create a file with intentional issues
cat > calculator.py << 'EOF'
def add(a, b):
    result = a + b
    return result

def divide(a, b):
    return a / b  # This has a bug!
EOF

git add calculator.py
```

```
git commit -m "Add calculator functions"
git push origin HEAD:refs/for/main
```

2. **Simulate review feedback:**
   In Gerrit UI, add these comments (or ask someone to):
   - "Line 7: Need to handle division by zero"
   - "Consider adding type hints for better code clarity"
   - "Add docstrings for the functions"

3. **Address the feedback:**

```
# Fix the issues
cat > calculator.py << 'EOF'
def add(a: int, b: int) -> int:
    """Add two numbers and return the result."""
    result = a + b
    return result

def divide(a: int, b: int) -> float:
    """Divide two numbers and return the result."""
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
EOF
```

4. **Update the change:**

```
git add calculator.py
git commit --amend
git push origin HEAD:refs/for/main
```

Success Criteria

- ☐ Change updated in Gerrit (same change number, new patch set)
- ☐ All feedback addressed
- ☐ Change ready for final review

Exercise 3: Review Someone Else's Change (Intermediate)

Goal

Learn how to properly review and test changes submitted by others.

Setup

Work with a partner or use a demo change.

Steps

1. **Find a change to review:**
   - Go to Gerrit UI
   - Find an "Active" change
   - Note the change number (e.g., 12345)
2. **Download the change:**

```
# Replace XX/YYYY/Z with actual change reference
git fetch origin refs/changes/45/12345/1 && git checkout FETCH_HEAD
```

3. **Test the change:**

```
# If it's code, run tests
python -m pytest  # or appropriate test command

# If it's a script, run it
python calculator.py

# Manual testing
# Try different inputs, edge cases, etc.
```

4. **Leave detailed feedback:**
   In Gerrit UI, add comments like:

```
"I tested this change with the following scenarios:

☑ Normal cases: add(2, 3) returns 5
☑ Edge cases: divide(10, 0) raises ValueError as expected
☑ All unit tests pass

Code looks good! One suggestion:
Line 12: Consider using f-strings for better readability

+1 (Looks good to me, but someone else must approve)"
```

5. **Return to your branch:**

```
git checkout main
```

Success Criteria

- ☐ Successfully downloaded and tested the change
- ☐ Left meaningful, constructive feedback
- ☐ Gave appropriate score (+1, +2, -1, or -2)

Exercise 4: Handle Merge Conflicts (Advanced)

Goal

Learn how to resolve conflicts when your change conflicts with others.

Steps

1. **Create two conflicting changes:**

   Change A (yours):

```
# Edit config.yml
echo "version: 1.0" > config.yml
echo "debug: true" >> config.yml

git add config.yml
git commit -m "Add debug configuration"
git push origin HEAD:refs/for/main
```

2. **Your change now conflicts (simulate):**

```
# Try to rebase
git fetch origin
git rebase origin/main

# Git will show conflicts in config.yml
```

3. **Resolve the conflicts:**

```
# Edit config.yml to merge both changes
cat > config.yml << 'EOF'
version: 1.0
debug: true
logging: enabled
EOF

git add config.yml
git rebase --continue
```

4. **Update your change:**

```
git push origin HEAD:refs/for/main
```

Success Criteria

- ☐ Successfully resolved merge conflicts
- ☐ Both features preserved in final version
- ☐ Change ready for review again

Self-Assessment Checklist

After completing these exercises, you should be able to:

Basic Skills

- ☐ Set up Gerrit hooks and configuration
- ☐ Submit changes for review
- ☐ Respond to review feedback
- ☐ Update changes with amendments

Intermediate Skills

- ☐ Review others' changes effectively
- ☐ Handle merge conflicts
- ☐ Work with dependent changes
- ☐ Use Gerrit UI efficiently

Advanced Skills

- ☐ Handle complex review scenarios
- ☐ Manage large refactorings
- ☐ Understand security implications
- ☐ Troubleshoot common issues

---

10. Quick Reference Card

Essential Commands Cheat Sheet

```
# Submit new change
git push origin HEAD:refs/for/main

# Update existing change
git commit --amend
git push origin HEAD:refs/for/main
```

```
# Download change for testing
git fetch origin refs/changes/XX/YYYY/Z && git checkout FETCH_HEAD

# Rebase before submit
git rebase origin/main
git push origin HEAD:refs/for/main

# Check your commits
git log --oneline -5
```

Review Scores Quick Guide

- **+2:** Approve (can merge)
- **+1:** Looks good (needs another +2)
- **-1:** Needs work (blocks merge)
- **-2:** Never merge (strong block)

Complete Command Reference

Repository Setup Commands

```
# Clone repository
git clone https://gerrit.example.com/myproject
cd myproject

# Install commit-msg hook
curl -Lo .git/hooks/commit-msg https://gerrit.example.com/tools/hooks/commit-
msg
chmod +x .git/hooks/commit-msg

# Configure Git (if needed)
git config user.name "Your Name"
git config user.email "your.email@example.com"
```

Working with Changes

```
# Submit to main branch
git push origin HEAD:refs/for/main

# Submit to specific branch
git push origin HEAD:refs/for/develop

# Submit as draft (private review)
git push origin HEAD:refs/drafts/main

# Submit with specific reviewers
```

```
git push origin
HEAD:refs/for/main%r=reviewer1@example.com,r=reviewer2@example.com

# Submit with topic
git push origin HEAD:refs/for/main%topic=feature-login
```

Update Changes

```
# Update existing change (amend commit)
git commit --amend
git push origin HEAD:refs/for/main

# Update with new commit message
git commit --amend -m "New commit message"
git push origin HEAD:refs/for/main

# Update specific files only
git add specific-file.txt
git commit --amend --no-edit
git push origin HEAD:refs/for/main
```

Download Changes

```
# Download specific change (patch set 1)
git fetch origin refs/changes/23/12345/1 && git checkout FETCH_HEAD

# Download latest patch set
git fetch origin refs/changes/23/12345/3 && git checkout FETCH_HEAD

# Download and create branch
git fetch origin refs/changes/23/12345/1
git checkout -b review-12345 FETCH_HEAD
```

Conflict Resolution

```
# Fetch latest changes
git fetch origin

# Rebase your change
git rebase origin/main

# During conflict resolution:
# Edit conflicted files
git add conflicted-file.txt
git rebase --continue
```

```
# Abort rebase if needed
git rebase --abort

# Force push after rebase
git push origin HEAD:refs/for/main
```

Change Information

```
# View commit history
git log --oneline -10

# View Change-Id in commits
git log --format="%h %s %b" -5

# Show specific commit details
git show HEAD

# View current branch
git branch

# Check git status
git status

# See what changed
git diff HEAD~1

# Compare with remote
git diff origin/main
```

Advanced Commands

```
# Cherry-pick specific change
git fetch origin refs/changes/23/12345/1
git cherry-pick FETCH_HEAD

# Submit multiple dependent changes
git push origin HEAD:refs/for/main
# Each commit becomes a separate change

# Reset to remote state
git reset --hard origin/main

# Clean untracked files
git clean -fd
```

```
# Generate Change-Id for existing commit
git commit --amend
```

Useful Git Aliases for Gerrit

Add these to your `.gitconfig` file:

```
[alias]
    # Gerrit-specific aliases
    gerrit-push = push origin HEAD:refs/for/main
    gerrit-draft = push origin HEAD:refs/drafts/main
    gerrit-rebase = !git fetch origin && git rebase origin/main

    # Common operations
    st = status
    co = checkout
    br = branch
    ci = commit
    amend = commit --amend

    # Log viewing
    lg = log --oneline --graph -10
    changes = log --format="%h %s %b" -5
```

Daily Workflow Commands

```
# 1. Get latest changes
git checkout main && git pull origin main

# 2. Make your changes
# ... edit files ...

# 3. Stage and commit
git add . && git commit -m "Your message"

# 4. Submit for review
git push origin HEAD:refs/for/main

# 5. Update after feedback
git commit --amend && git push origin HEAD:refs/for/main
```