Chapter 2: Basic Installation and Setup

Overview

In this chapter, we'll install Gerrit step by step. We'll start with a simple local installation that you can use for learning and testing.

System Requirements

Minimum Requirements

- **RAM**: 4GB (8GB recommended)
- **Storage**: 2GB free space (more for actual projects)
- **Java**: OpenJDK 11 or newer
- **Operating System**: Windows, Linux, or macOS

What We'll Install

1. **Java Development Kit (JDK)**
2. **Gerrit Code Review**
3. **Git** (if not already installed)

Step 1: Install Java

Gerrit requires Java to run. Let's check if you already have it and install if needed.

Check Existing Java Installation

Open your command prompt/terminal and run:

Windows PowerShell:

```
java -version
```

Expected Output (if Java is installed):

```
openjdk version "11.0.16" 2022-07-19
OpenJDK Runtime Environment (build 11.0.16+8-post-Ubuntu-0ubuntu120.04)
OpenJDK 64-Bit Server VM (build 11.0.16+8-post-Ubuntu-0ubuntu120.04, mixed
mode, sharing)
```

Install Java (if needed)

Windows

1. **Download OpenJDK 11+**
    - Visit: https://adoptium.net/
    - Download "Eclipse Temurin" for Windows
    - Choose the latest LTS version (11, 17, or 21)
2. **Install**
    - Run the downloaded installer
    - Follow the installation wizard
    - ☑ Check "Set JAVA_HOME variable"
    - ☑ Check "Add to PATH"
3. **Verify Installation**

```
java -version
javac -version
echo $env:JAVA_HOME
```

Linux (Ubuntu/Debian)

```
# Update package list
sudo apt update

# Install OpenJDK 11
sudo apt install openjdk-11-jdk

# Verify installation
java -version
javac -version
```

macOS

```
# Using Homebrew (install Homebrew first if needed)
brew install openjdk@11

# Add to PATH (add this to your ~/.zshrc or ~/.bash_profile)
export PATH="/opt/homebrew/opt/openjdk@11/bin:$PATH"

# Verify installation
java -version
```

Step 2: Download Gerrit

Download the Latest Version

1. **Visit Gerrit Downloads**
   - Go to: https://gerrit-releases.storage.googleapis.com/index.html
   - Find the latest stable release (e.g., 3.8.x)
   - wget https://gerrit-releases.storage.googleapis.com/gerrit-3.12.1.war
2. **Download for Your System**
   - Look for files ending in `.war` (Web Application Archive)
   - Example: `gerrit-3.8.0.war`
3. **Create Gerrit Directory**

   Windows:

```
# Create directory for Gerrit
mkdir C:\gerrit
cd C:\gerrit

# Move downloaded file here
# (Download to this folder or move it here)
```

   Linux/macOS:

```
# Create directory for Gerrit
mkdir ~/gerrit
cd ~/gerrit

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

Step 3: Initialize Gerrit

Basic Initialization

Let's set up Gerrit with basic configuration:

Windows:

```
cd C:\gerrit

# Initialize Gerrit (this will take a few minutes)
java -jar gerrit-3.8.0.war init -d gerrit_site
```

Linux/macOS:

```
cd ~/gerrit

# Initialize Gerrit
java -jar gerrit-3.8.0.war init -d gerrit_site
```

Configuration Wizard

During initialization, Gerrit will ask several questions. Here are the recommended answers for beginners:

```
*** Gerrit Code Review 3.8.0
***

Create '/path/to/gerrit_site' [Y/n]? Y

*** Git Repositories
***

Location of Git repositories   [git]: ENTER (use default)

*** Database
***

Database server type           [h2]: ENTER (use default H2 database)

*** Index
***

Type                           [lucene]: ENTER (use default)

*** User Authentication
***

Authentication method          [development_become_any_account]: ENTER (for
learning)

*** Review Labels
***

Install Verified label         [y]: Y

*** Email Delivery
***

SMTP server hostname           [localhost]: ENTER (skip for now)
SMTP server port               [(default)]: ENTER
SMTP encryption                [none]: ENTER
SMTP username                  : ENTER (leave blank)
```

```
*** Container Process
***

Run as                        [your_username]: ENTER
Java runtime                  [/path/to/java]: ENTER (use detected Java)
Copy gerrit-3.8.0.war to gerrit_site/bin/gerrit.war [Y/n]? Y
Copying gerrit-3.8.0.war to gerrit_site/bin/gerrit.war

*** SSH Daemon
***

Listen on address             [*]: ENTER
Listen on port                [29418]: ENTER

*** HTTP Daemon
***

Behind reverse proxy          [y/N]? N
Use SSL (https://)            [y/N]? N (we'll configure this later)
Listen on address             [*]: ENTER
Listen on port                [8080]: ENTER
Canonical URL                 [http://localhost:8080/]: ENTER

*** Cache
***

*** Plugins
***

Install plugin codemirror-editor version v3.8.0 [y/N]? Y
Install plugin commit-message-length-validator version v3.8.0 [y/N]? Y
Install plugin download-commands version v3.8.0 [y/N]? Y
Install plugin replication version v3.8.0 [y/N]? N (for now)
Install plugin reviewnotes version v3.8.0 [y/N]? Y

Initialized /path/to/gerrit_site
```

Step 4: Start Gerrit

Start the Gerrit Service

Windows:

```
cd C:\gerrit\gerrit_site

# Start Gerrit
.\bin\gerrit.sh start
```

Linux/macOS:

```
cd ~/gerrit/gerrit_site

# Start Gerrit
./bin/gerrit.sh start
```

Verify Gerrit is Running

1. **Check the Process**

   ```
   # Linux/macOS
   ./bin/gerrit.sh check

   # Should show: Gerrit running pid=XXXX
   ```

2. **Open Web Interface**
   - Open your web browser
   - Go to: http://localhost:8080
   - You should see the Gerrit welcome page!

Step 5: Create Your First Admin User

Access Gerrit Web Interface

1. **Open Browser**
   - Navigate to http://localhost:8080
2. **Become Admin (Development Mode)**
   - Click "Become" in the top-right corner
   - Enter your desired username (e.g., "admin")
   - Enter your full name
   - Enter your email address
3. **Verify Admin Access**
   - You should now see admin options in the interface
   - Look for "Admin" menu items

Understanding the Directory Structure

After initialization, your Gerrit directory looks like this:

```

```

```
gerrit_site/
├── bin/
│   ├── gerrit.sh          # Start/stop script
│   └── gerrit.war         # Gerrit application
├── cache/                 # Cache files
├── data/                  # Database files
├── etc/
│   ├── gerrit.config      # Main configuration
│   ├── secure.config      # Sensitive settings
│   └── ...
├── git/                   # Git repositories
├── index/                 # Search index
├── lib/                   # Additional libraries
├── logs/                  # Log files
├── plugins/               # Installed plugins
└── static/                # Web assets
```

Key Configuration Files

gerrit.config

This is the main configuration file:

```
[gerrit]
    basePath = git
    serverId = 12345678-1234-1234-1234-123456789012
    canonicalWebUrl = http://localhost:8080/

[database]
    type = h2
    database = /path/to/gerrit_site/db/ReviewDB

[auth]
    type = DEVELOPMENT_BECOME_ANY_ACCOUNT

[receive]
    enableSignedPush = false

[sendemail]
    smtpServer = localhost

[sshd]
    listenAddress = *:29418

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

[cache]
    directory = cache
```

Step 6: Configure HTTPS (SSL/TLS)

For production environments and enhanced security, you should configure HTTPS. We'll cover two methods: Let's Encrypt (free, automated) and self-signed certificates (for testing/internal use).

Method 1: Let's Encrypt (Recommended for Production)

Let's Encrypt provides free, automated SSL certificates that are trusted by all browsers.

Prerequisites for Let's Encrypt

- **Domain name** pointing to your server (e.g., gerrit.yourcompany.com)
- **Port 80 and 443** accessible from the internet
- **Root/sudo access** on the server

Install Certbot

Ubuntu/Debian:

```
sudo apt update
sudo apt install certbot nginx
```

CentOS/RHEL:

```
sudo yum install epel-release
sudo yum install certbot nginx
```

Windows:
For Windows, we'll use a different approach with win-acme or manual certificate generation.

Option A: Direct Let's Encrypt Integration

Step 1: Stop Gerrit temporarily

```
cd ~/gerrit/gerrit_site
./bin/gerrit.sh stop
```

Step 2: Generate certificate

```
# Replace gerrit.yourcompany.com with your actual domain
sudo certbot certonly --standalone -d gerrit.yourcompany.com
```

```
# Certificate files will be saved to:
# /etc/letsencrypt/live/gerrit.yourcompany.com/
```

Step 3: Convert certificates for Java

```
# Create PKCS12 keystore from Let's Encrypt certificates
sudo openssl pkcs12 -export \
    -in /etc/letsencrypt/live/gerrit.yourcompany.com/fullchain.pem \
    -inkey /etc/letsencrypt/live/gerrit.yourcompany.com/privkey.pem \
    -out /opt/gerrit/gerrit_site/etc/keystore.p12 \
    -name gerrit \
    -password pass:changeit

# Convert to JKS format
sudo keytool -importkeystore \
    -srckeystore /opt/gerrit/gerrit_site/etc/keystore.p12 \
    -srcstoretype PKCS12 \
    -srcstorepass changeit \
    -destkeystore /opt/gerrit/gerrit_site/etc/keystore.jks \
    -deststoretype JKS \
    -deststorepass changeit

# Set proper ownership
sudo chown gerrit:gerrit /opt/gerrit/gerrit_site/etc/keystore.*
```

Step 4: Configure Gerrit for HTTPS

```
# Edit gerrit.config
nano ~/gerrit/gerrit_site/etc/gerrit.config
```

Update the configuration:

```
[gerrit]
    basePath = git
    serverId = 12345678-1234-1234-1234-123456789012
    canonicalWebUrl = https://gerrit.yourcompany.com/

[httpd]
    listenUrl = https://*:8443/
    sslKeyStore = etc/keystore.jks
    sslKeyStorePassword = changeit
    sslTrustStore = etc/keystore.jks
    sslTrustStorePassword = changeit

# Optional: Redirect HTTP to HTTPS
```

```
[httpd]
    filterClass = com.googlesource.gerrit.httpd.raw.StaticModule$GuiceFilter
```

Step 5: Create certificate renewal script

```
# Create renewal script
sudo tee /opt/gerrit/renew-cert.sh > /dev/null << 'EOF'
#!/bin/bash
# Let's Encrypt certificate renewal for Gerrit

DOMAIN="gerrit.yourcompany.com"
GERRIT_SITE="/opt/gerrit/gerrit_site"

# Renew certificate
certbot renew --quiet

# Convert to Java keystore format
openssl pkcs12 -export \
    -in /etc/letsencrypt/live/$DOMAIN/fullchain.pem \
    -inkey /etc/letsencrypt/live/$DOMAIN/privkey.pem \
    -out $GERRIT_SITE/etc/keystore.p12 \
    -name gerrit \
    -password pass:changeit

keytool -importkeystore \
    -srckeystore $GERRIT_SITE/etc/keystore.p12 \
    -srcstoretype PKCS12 \
    -srcstorepass changeit \
    -destkeystore $GERRIT_SITE/etc/keystore.jks \
    -deststoretype JKS \
    -deststorepass changeit \
    -noprompt

# Set proper ownership
chown gerrit:gerrit $GERRIT_SITE/etc/keystore.*

# Restart Gerrit to load new certificate
systemctl restart gerrit

echo "Certificate renewed and Gerrit restarted"
EOF

chmod +x /opt/gerrit/renew-cert.sh
```

Step 6: Setup automatic renewal

```
# Add to crontab for automatic renewal
sudo crontab -e
```

```
# Add this line (runs at 2 AM daily)
0 2 * * * /opt/gerrit/renew-cert.sh
```

Option B: Reverse Proxy with Let's Encrypt (Recommended)

This approach uses Nginx as a reverse proxy to handle SSL termination.

Step 1: Install and configure Nginx

```
sudo apt install nginx

# Create Nginx configuration for Gerrit
sudo tee /etc/nginx/sites-available/gerrit << 'EOF'
server {
    listen 80;
    server_name gerrit.yourcompany.com;

    # Redirect HTTP to HTTPS
    return 301 https://$server_name$request_uri;
}

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

    # SSL configuration (will be added by Certbot)

    # Proxy settings
    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;

        # WebSocket support for live updates
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        # Timeouts
        proxy_connect_timeout 30s;
        proxy_send_timeout 30s;
        proxy_read_timeout 30s;
    }

    # Static content optimization
```

```
    location ~ ^/(static|Documentation)/ {
        proxy_pass http://127.0.0.1:8080;
        proxy_cache_valid 200 1d;
        expires 1d;
        add_header Cache-Control "public, immutable";
    }

    # Security headers
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains"
always;
}
EOF

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

Step 2: Get Let's Encrypt certificate with Nginx

```
# Get certificate (Nginx plugin handles the configuration)
sudo certbot --nginx -d gerrit.yourcompany.com

# Test automatic renewal
sudo certbot renew --dry-run
```

Step 3: Configure Gerrit for reverse proxy

```
# Edit gerrit.config
[gerrit]
    canonicalWebUrl = https://gerrit.yourcompany.com/

[httpd]
    listenUrl = proxy-https://127.0.0.1:8080/

[auth]
    # If using authentication
    trustContainerAuth = true
```

Method 2: Self-Signed Certificates (Development/Internal Use)

For development or internal environments where you don't need publicly trusted certificates.

Generate Self-Signed Certificate

Step 1: Create certificate

```
# Create directory for certificates
mkdir -p ~/gerrit/gerrit_site/etc/ssl

cd ~/gerrit/gerrit_site/etc/ssl

# Generate private key
openssl genrsa -out gerrit.key 2048

# Generate certificate signing request
openssl req -new -key gerrit.key -out gerrit.csr

# You'll be prompted for information:
Country Name (2 letter code) [AU]: US
State or Province Name (full name) [Some-State]: Your State
Locality Name (eg, city) []: Your City
Organization Name (eg, company) [Internet Widgits Pty Ltd]: Your Company
Organizational Unit Name (eg, section) []: IT Department
Common Name (e.g. server FQDN or YOUR name) []: gerrit.local
Email Address []: admin@yourcompany.com

# Generate self-signed certificate (valid for 1 year)
openssl x509 -req -days 365 -in gerrit.csr -signkey gerrit.key -out gerrit.crt

# Create Java keystore
openssl pkcs12 -export -in gerrit.crt -inkey gerrit.key -out gerrit.p12 -name
gerrit -password pass:changeit

keytool -importkeystore \
    -srckeystore gerrit.p12 \
    -srcstoretype PKCS12 \
    -srcstorepass changeit \
    -destkeystore gerrit.jks \
    -deststoretype JKS \
    -deststorepass changeit
```

Advanced Self-Signed Certificate with SAN

For a more robust self-signed certificate with Subject Alternative Names:

```
# Create configuration file for certificate
cat > gerrit-ssl.conf << 'EOF'
[req]
default_bits = 2048
prompt = no
default_md = sha256
distinguished_name = dn
req_extensions = v3_req
```

```
[dn]
C = US
ST = Your State
L = Your City
O = Your Organization
OU = IT Department
CN = gerrit.local

[v3_req]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = gerrit.local
DNS.2 = gerrit.yourcompany.com
DNS.3 = localhost
IP.1 = 127.0.0.1
IP.2 = 192.168.1.100
EOF

# Generate certificate with SAN
openssl req -new -x509 -days 365 -nodes \
    -out gerrit.crt \
    -keyout gerrit.key \
    -config gerrit-ssl.conf \
    -extensions v3_req

# Convert to Java keystore
openssl pkcs12 -export -in gerrit.crt -inkey gerrit.key \
    -out gerrit.p12 -name gerrit -password pass:changeit

keytool -importkeystore \
    -srckeystore gerrit.p12 \
    -srcstoretype PKCS12 \
    -srcstorepass changeit \
    -destkeystore gerrit.jks \
    -deststoretype JKS \
    -deststorepass changeit
```

Configure Gerrit for Self-Signed SSL

Edit gerrit.config:

```
[gerrit]
    basePath = git
    serverId = 12345678-1234-1234-1234-123456789012
    canonicalWebUrl = https://gerrit.local:8443/
```

```
[httpd]
    listenUrl = https://*:8443/
    sslKeyStore = etc/ssl/gerrit.jks
    sslKeyStorePassword = changeit
    sslTrustStore = etc/ssl/gerrit.jks
    sslTrustStorePassword = changeit
```

Windows HTTPS Configuration

For Windows environments, here's a PowerShell script to set up self-signed certificates:

```
# windows-ssl-setup.ps1

# Create SSL directory
New-Item -ItemType Directory -Force -Path "C:\gerrit\gerrit_site\etc\ssl"
Set-Location "C:\gerrit\gerrit_site\etc\ssl"

# Generate self-signed certificate using PowerShell
$cert = New-SelfSignedCertificate -DnsName "gerrit.local", "localhost" -
CertStoreLocation "cert:\LocalMachine\My" -KeyLength 2048 -KeyAlgorithm RSA -
HashAlgorithm SHA256 -KeyUsage KeyEncipherment,DigitalSignature -Type
SSLServerAuthentication -NotAfter (Get-Date).AddYears(1)

# Export certificate to PFX
$certPassword = ConvertTo-SecureString -String "changeit" -Force -AsPlainText
Export-PfxCertificate -Cert $cert -FilePath "gerrit.pfx" -Password
$certPassword

# Convert PFX to JKS using Java keytool
& "keytool" -importkeystore -srckeystore "gerrit.pfx" -srcstoretype PKCS12 -
srcstorepass "changeit" -destkeystore "gerrit.jks" -deststoretype JKS -
deststorepass "changeit"

Write-Host "SSL certificate generated successfully!"
Write-Host "Certificate file: gerrit.jks"
Write-Host "Password: changeit"
```

SSL Configuration Scripts

Here's a comprehensive script to automate SSL setup:

```
#!/bin/bash
# ssl-setup.sh - Automated SSL setup for Gerrit

set -e

DOMAIN=""
EMAIL=""
```

```
METHOD=""
GERRIT_SITE="/opt/gerrit/gerrit_site"

usage() {
    echo "Usage: $0 -d domain -e email -m method"
    echo "Methods: letsencrypt, selfsigned, letsencrypt-nginx"
    echo "Example: $0 -d gerrit.company.com -e admin@company.com -m
letsencrypt"
    exit 1
}

while getopts "d:e:m:" opt; do
    case $opt in
        d) DOMAIN="$OPTARG" ;;
        e) EMAIL="$OPTARG" ;;
        m) METHOD="$OPTARG" ;;
        *) usage ;;
    esac
done

if [ -z "$DOMAIN" ] || [ -z "$METHOD" ]; then
    usage
fi

setup_letsencrypt_direct() {
    echo "Setting up Let's Encrypt direct integration..."

    # Stop Gerrit
    $GERRIT_SITE/bin/gerrit.sh stop

    # Get certificate
    certbot certonly --standalone -d $DOMAIN --email $EMAIL --agree-tos --non-
interactive

    # Convert to Java keystore
    openssl pkcs12 -export \
        -in /etc/letsencrypt/live/$DOMAIN/fullchain.pem \
        -inkey /etc/letsencrypt/live/$DOMAIN/privkey.pem \
        -out $GERRIT_SITE/etc/keystore.p12 \
        -name gerrit \
        -password pass:changeit

    keytool -importkeystore \
        -srckeystore $GERRIT_SITE/etc/keystore.p12 \
        -srcstoretype PKCS12 \
        -srcstorepass changeit \
        -destkeystore $GERRIT_SITE/etc/keystore.jks \
        -deststoretype JKS \
        -deststorepass changeit \
        -noprompt

    # Update Gerrit configuration
    sed -i "s|canonicalWebUrl = .*|canonicalWebUrl = https://$DOMAIN/|"
```

```
$GERRIT_SITE/etc/gerrit.config
    sed -i "s|listenUrl = .*|listenUrl = https://*:8443/|"
$GERRIT_SITE/etc/gerrit.config

    # Add SSL configuration
    if ! grep -q "sslKeyStore" $GERRIT_SITE/etc/gerrit.config; then
        cat >> $GERRIT_SITE/etc/gerrit.config << EOF

[httpd]
    sslKeyStore = etc/keystore.jks
    sslKeyStorePassword = changeit
EOF
    fi

    chown -R gerrit:gerrit $GERRIT_SITE/etc/keystore.*

    # Start Gerrit
    $GERRIT_SITE/bin/gerrit.sh start

    echo "Let's Encrypt SSL configured successfully!"
    echo "Access Gerrit at: https://$DOMAIN"
}

setup_selfsigned() {
    echo "Setting up self-signed certificate..."

    mkdir -p $GERRIT_SITE/etc/ssl
    cd $GERRIT_SITE/etc/ssl

    # Create certificate configuration
    cat > ssl.conf << EOF
[req]
default_bits = 2048
prompt = no
default_md = sha256
distinguished_name = dn
req_extensions = v3_req

[dn]
C = US
ST = State
L = City
O = Organization
OU = IT Department
CN = $DOMAIN

[v3_req]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
DNS.1 = $DOMAIN
```

```
DNS.2 = localhost
IP.1 = 127.0.0.1
EOF

    # Generate certificate
    openssl req -new -x509 -days 365 -nodes \
        -out gerrit.crt \
        -keyout gerrit.key \
        -config ssl.conf \
        -extensions v3_req

    # Convert to Java keystore
    openssl pkcs12 -export -in gerrit.crt -inkey gerrit.key \
        -out gerrit.p12 -name gerrit -password pass:changeit

    keytool -importkeystore \
        -srckeystore gerrit.p12 \
        -srcstoretype PKCS12 \
        -srcstorepass changeit \
        -destkeystore gerrit.jks \
        -deststoretype JKS \
        -deststorepass changeit \
        -noprompt

    # Update Gerrit configuration
    sed -i "s|canonicalWebUrl = .*|canonicalWebUrl = https://$DOMAIN:8443/|"
$GERRIT_SITE/etc/gerrit.config
    sed -i "s|listenUrl = .*|listenUrl = https://*:8443/|"
$GERRIT_SITE/etc/gerrit.config

    # Add SSL configuration
    if ! grep -q "sslKeyStore" $GERRIT_SITE/etc/gerrit.config; then
        cat >> $GERRIT_SITE/etc/gerrit.config << EOF

[httpd]
    sslKeyStore = etc/ssl/gerrit.jks
    sslKeyStorePassword = changeit
EOF
    fi

    chown -R gerrit:gerrit $GERRIT_SITE/etc/ssl/

    # Restart Gerrit
    $GERRIT_SITE/bin/gerrit.sh restart

    echo "Self-signed SSL configured successfully!"
    echo "Access Gerrit at: https://$DOMAIN:8443"
    echo "Note: You'll need to accept the security warning in your browser"
}

setup_letsencrypt_nginx() {
    echo "Setting up Let's Encrypt with Nginx reverse proxy..."
```

```
    # Install Nginx if not present
    if ! command -v nginx &> /dev/null; then
        apt update && apt install -y nginx
    fi

    # Create Nginx configuration
    cat > /etc/nginx/sites-available/gerrit << EOF
server {
    listen 80;
    server_name $DOMAIN;
    return 301 https://\$server_name\$request_uri;
}

server {
    listen 443 ssl http2;
    server_name $DOMAIN;

    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_http_version 1.1;
        proxy_set_header Upgrade \$http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains"
always;
}
EOF

    ln -sf /etc/nginx/sites-available/gerrit /etc/nginx/sites-enabled/
    nginx -t && systemctl reload nginx

    # Get Let's Encrypt certificate
    certbot --nginx -d $DOMAIN --email $EMAIL --agree-tos --non-interactive

    # Update Gerrit for reverse proxy
    sed -i "s|canonicalWebUrl = .*|canonicalWebUrl = https://$DOMAIN/|"
$GERRIT_SITE/etc/gerrit.config
    sed -i "s|listenUrl = .*|listenUrl = proxy-https://127.0.0.1:8080/|"
$GERRIT_SITE/etc/gerrit.config

    $GERRIT_SITE/bin/gerrit.sh restart

    echo "Let's Encrypt with Nginx configured successfully!"
    echo "Access Gerrit at: https://$DOMAIN"
}
```

```
case $METHOD in
    "letsencrypt")
        setup_letsencrypt_direct
        ;;
    "selfsigned")
        setup_selfsigned
        ;;
    "letsencrypt-nginx")
        setup_letsencrypt_nginx
        ;;
    *)
        echo "Unknown method: $METHOD"
        usage
        ;;
esac

echo "SSL setup completed!"
```

Testing SSL Configuration

After configuring SSL, test your setup:

```
#!/bin/bash
# test-ssl.sh

DOMAIN="$1"
PORT="${2:-443}"

if [ -z "$DOMAIN" ]; then
    echo "Usage: $0 <domain> [port]"
    exit 1
fi

echo "Testing SSL configuration for $DOMAIN:$PORT"
echo "============================================="

# Test SSL certificate
echo "1. Certificate information:"
openssl s_client -connect $DOMAIN:$PORT -servername $DOMAIN < /dev/null
2>/dev/null | openssl x509 -noout -text | grep -E "(Subject:|Issuer:|Not
Before:|Not After :|DNS:|IP Address:)"

echo ""
echo "2. SSL Labs style check:"
# Check cipher suites
echo "Supported cipher suites:"
nmap --script ssl-enum-ciphers -p $PORT $DOMAIN 2>/dev/null | grep -E "
(TLS|SSL)"

echo ""
```

```
echo "3. Certificate chain verification:"
openssl s_client -connect $DOMAIN:$PORT -verify_return_error < /dev/null

echo ""
echo "4. HTTPS response test:"
curl -I https://$DOMAIN 2>/dev/null | head -5

echo ""
echo "SSL test completed!"
```

Troubleshooting SSL Issues

Common SSL problems and solutions:

Issue 1: Certificate Not Trusted

Symptoms: Browser shows "Not Secure" or certificate warnings
Solutions:

- For Let's Encrypt: Ensure domain points to server and port 80/443 are accessible
- For self-signed: Add certificate to browser/system trust store
- Check certificate chain completeness

Issue 2: Mixed Content Warnings

Symptoms: Some resources load over HTTP instead of HTTPS
Solution:

```
# In gerrit.config, ensure canonical URL uses HTTPS
[gerrit]
    canonicalWebUrl = https://your-domain.com/

# For reverse proxy setups
[httpd]
    listenUrl = proxy-https://127.0.0.1:8080/
```

Issue 3: SSL Handshake Failures

Symptoms: Connection timeouts or SSL handshake errors
Solutions:

```
# Check if SSL port is accessible
telnet your-domain.com 443

# Test SSL handshake
openssl s_client -connect your-domain.com:443
```

```
# Check Gerrit logs
tail -f logs/error_log | grep -i ssl
```

Issue 4: Certificate Renewal Issues

Solution for Let's Encrypt:

```
# Test renewal
certbot renew --dry-run

# Check renewal logs
grep renewal /var/log/letsencrypt/letsencrypt.log

# Manual renewal if needed
certbot renew --force-renewal
```

Now let's continue with the existing content...

Create a Test Project

1. **Access Gerrit Web UI** (http://localhost:8080)
2. **Create New Project**
   - Click "Admin" → "Projects"
   - Click "Create New Project"
   - Project Name: `test-project`
   - Check "Create initial empty commit"
   - Click "Create Project"
3. **Clone the Project**

```
# Windows
cd C:\
git clone http://localhost:8080/test-project
cd test-project

# Configure git user
git config user.name "Your Name"
git config user.email "your.email@example.com"
```

Install Commit-msg Hook

This is crucial for Gerrit to work properly:

```
# Windows (in your project directory)
scp -p -P 29418 admin@localhost:hooks/commit-msg .git/hooks/
```

If the above doesn't work (common on Windows), manually download:

1. Go to http://localhost:8080/tools/hooks/commit-msg
2. Save the file to your project's `.git/hooks/` directory
3. Make it executable (if on Linux/macOS): `chmod +x .git/hooks/commit-msg`

Common Installation Issues

Issue 1: Java Not Found

Error: `java: command not found`
Solution:

- Verify Java installation
- Check PATH environment variable
- Restart terminal after Java installation

Issue 2: Permission Denied

Error: Permission denied when starting Gerrit
Solution:

```
# Linux/macOS - make script executable
chmod +x bin/gerrit.sh

# Or run with explicit permissions
sudo ./bin/gerrit.sh start
```

Issue 3: Port Already in Use

Error: `Address already in use: bind`
Solution:

- Check what's using port 8080: `netstat -ano | findstr :8080` (Windows)
- Change port in `etc/gerrit.config`
- Or stop the conflicting service

Issue 4: Web Interface Not Loading

Checklist:

- Is Gerrit actually running? (`./bin/gerrit.sh check`)
- Is the URL correct? (http://localhost:8080)
- Check firewall settings

- Review logs: `tail -f logs/error_log`

Step 7: Testing Your Installation

Basic HTTP Testing

Let's verify that your basic Gerrit installation is working correctly:

1. Service Status Check

```
# Windows - Check if Gerrit is running
cd C:\gerrit\gerrit_site
.\bin\gerrit.sh check

# Expected output: Gerrit running pid=XXXX
```

```
# Linux/macOS - Check Gerrit status
cd ~/gerrit/gerrit_site
./bin/gerrit.sh check

# Additional process check
ps aux | grep gerrit
```

2. Port Accessibility Test

```
# Windows - Test if port 8080 is accessible
Test-NetConnection -ComputerName localhost -Port 8080

# Alternative using telnet
telnet localhost 8080
```

```
# Linux/macOS - Test port connectivity
telnet localhost 8080

# Or using netcat
nc -zv localhost 8080

# Check listening ports
netstat -tulpn | grep :8080
```

3. HTTP Response Test

```
# Windows PowerShell - Test HTTP response
Invoke-WebRequest -Uri "http://localhost:8080" -Method HEAD

# Using curl (if available)
curl -I http://localhost:8080
```

```
# Linux/macOS - Test HTTP response
curl -I http://localhost:8080

# Expected output should include:
# HTTP/1.1 200 OK
# Content-Type: text/html; charset=UTF-8
```

4. Web Interface Verification

- Open browser and navigate to http://localhost:8080
- You should see the Gerrit login page
- Click "Become" to enter as admin
- Verify you can access Admin panels

HTTPS/SSL Testing

If you configured SSL, perform these additional tests:

1. SSL Certificate Verification

```
# Test SSL certificate info
openssl s_client -connect localhost:8443 -servername localhost < /dev/null
2>/dev/null | openssl x509 -noout -text

# Quick certificate details
echo | openssl s_client -connect localhost:8443 2>/dev/null | openssl x509 -
noout -dates -subject -issuer
```

2. SSL Connectivity Test

```
# Windows - Test HTTPS connectivity
Test-NetConnection -ComputerName localhost -Port 8443

# Test SSL handshake
openssl s_client -connect localhost:8443
```

```
# Linux/macOS - Test SSL port
nc -zv localhost 8443

# Test SSL handshake with details
openssl s_client -connect localhost:8443 -verify_return_error
```

3. HTTPS Response Test

```
# Windows - Test HTTPS response (may show certificate warnings)
Invoke-WebRequest -Uri "https://localhost:8443" -Method HEAD -
SkipCertificateCheck

# Using curl
curl -I -k https://localhost:8443
```

```
# Linux/macOS - Test HTTPS response
curl -I -k https://localhost:8443

# For Let's Encrypt certificates (should be trusted)
curl -I https://your-domain.com

# Check certificate chain
curl -v https://your-domain.com 2>&1 | grep -E "(certificate|SSL)"
```

4. SSL Configuration Validation

```
# Comprehensive SSL test script
create_ssl_test() {
    local domain="$1"
    local port="${2:-443}"

    echo "=== SSL Configuration Test for $domain:$port ==="

    # Test 1: Basic connectivity
    echo "1. Testing connectivity..."
    if nc -zv $domain $port 2>&1; then
        echo "✅ Port $port is accessible"
    else
        echo "❌ Cannot connect to port $port"
        return 1
    fi

    # Test 2: Certificate information
    echo -e "\n2. Certificate information:"
    openssl s_client -connect $domain:$port -servername $domain < /dev/null
```

```
2>/dev/null | \
    openssl x509 -noout -text | grep -E "(Subject:|Issuer:|Not Before:|Not
After:|DNS:|IP Address:)"

    # Test 3: Certificate expiration
    echo -e "\n3. Certificate validity:"
    openssl s_client -connect $domain:$port -servername $domain < /dev/null
2>/dev/null | \
    openssl x509 -noout -dates

    # Test 4: SSL/TLS version support
    echo -e "\n4. Supported TLS versions:"
    for version in ssl3 tls1 tls1_1 tls1_2 tls1_3; do
        if openssl s_client -connect $domain:$port -$version < /dev/null
2>/dev/null | grep -q "Verify return code: 0"; then
            echo "✅ $version supported"
        else
            echo "❌ $version not supported/available"
        fi
    done

    # Test 5: Cipher suites
    echo -e "\n5. Strong cipher suites:"
    openssl s_client -connect $domain:$port -cipher
'ECDHE+AESGCM:ECDHE+CHACHA20:DHE+AESGCM:DHE+CHACHA20:!aNULL:!MD5:!DSS' <
/dev/null 2>/dev/null | \
    grep -E "(Cipher|Protocol)"

    echo -e "\n✅ SSL test completed for $domain:$port"
}

# Usage examples:
# create_ssl_test localhost 8443
# create_ssl_test gerrit.yourcompany.com 443
```

Database and Storage Testing

1. Database Connectivity

```
# Check H2 database files
ls -la ~/gerrit/gerrit_site/db/

# For production setups, test PostgreSQL connectivity
psql -h localhost -U gerrit_user -d gerrit_db -c "SELECT version();"
```

2. Git Repository Access

```
# Test Git repository storage
ls -la ~/gerrit/gerrit_site/git/

# Test Git operations (after creating a test project)
git ls-remote http://localhost:8080/test-project
```

3. Log File Monitoring

```
# Monitor Gerrit logs for errors
tail -f ~/gerrit/gerrit_site/logs/error_log

# Check for startup issues
grep -i error ~/gerrit/gerrit_site/logs/error_log | tail -10

# Monitor access logs
tail -f ~/gerrit/gerrit_site/logs/httpd_log
```

Performance Testing

1. Memory Usage Check

```
# Check Gerrit Java process memory usage
ps aux | grep gerrit | grep -v grep

# More detailed memory info (Linux)
cat /proc/$(pgrep -f gerrit)/status | grep -E "(VmSize|VmRSS|VmData)"
```

```
# Windows - Check memory usage
Get-Process | Where-Object {$_.ProcessName -like "*java*"} | Select-Object
ProcessName, WorkingSet, CPU
```

2. Response Time Testing

```
# Simple response time test
time curl -s http://localhost:8080 > /dev/null

# More detailed timing
curl -w "@-" -s http://localhost:8080 << 'EOF'
     time_namelookup:  %{time_namelookup}\n
        time_connect:  %{time_connect}\n
     time_appconnect:  %{time_appconnect}\n
    time_pretransfer:  %{time_pretransfer}\n
       time_redirect:  %{time_redirect}\n
```

```
  time_starttransfer:  %{time_starttransfer}\n
                     ----------\n
          time_total:  %{time_total}\n
EOF
```

3. Load Testing (Optional)

```
# Install Apache Bench (ab) for basic load testing
# Ubuntu/Debian: sudo apt install apache2-utils
# CentOS/RHEL: sudo yum install httpd-tools

# Simple load test (10 concurrent users, 100 requests)
ab -n 100 -c 10 http://localhost:8080/

# Monitor during load test
watch 'ps aux | grep gerrit | grep -v grep'
```

SSH Access Testing

1. SSH Port Test

```
# Test SSH connectivity (port 29418)
telnet localhost 29418

# Test SSH with key (after setting up SSH keys)
ssh -p 29418 admin@localhost gerrit version
```

2. SSH Key Setup Verification

```
# Generate SSH key (if not exists)
ssh-keygen -t rsa -b 4096 -C "your-email@example.com"

# Add public key to Gerrit through web interface
# Settings → SSH Public Keys → Add Key

# Test SSH connection
ssh -p 29418 admin@localhost help
```

Complete Installation Test Script

Here's a comprehensive test script that checks everything:

```
#!/bin/bash
# test-gerrit-installation.sh - Comprehensive Gerrit installation test
```

```
set -e

GERRIT_HOST="${1:-localhost}"
HTTP_PORT="${2:-8080}"
HTTPS_PORT="${3:-8443}"
SSH_PORT="${4:-29418}"
GERRIT_SITE="${5:-/opt/gerrit/gerrit_site}"

echo "🩹 Testing Gerrit Installation"
echo "================================"
echo "Host: $GERRIT_HOST"
echo "HTTP Port: $HTTP_PORT"
echo "HTTPS Port: $HTTPS_PORT"
echo "SSH Port: $SSH_PORT"
echo "Site: $GERRIT_SITE"
echo ""

# Test 1: Service Status
echo "1️⃣  Testing service status..."
if [ -f "$GERRIT_SITE/bin/gerrit.sh" ]; then
    if $GERRIT_SITE/bin/gerrit.sh check > /dev/null 2>&1; then
        echo "✅ Gerrit service is running"
    else
        echo "❌ Gerrit service is not running"
        exit 1
    fi
else
    echo "❌ Gerrit installation not found at $GERRIT_SITE"
    exit 1
fi

# Test 2: HTTP Connectivity
echo -e "\n2️⃣  Testing HTTP connectivity..."
if curl -s -f "http://$GERRIT_HOST:$HTTP_PORT" > /dev/null; then
    echo "✅ HTTP interface accessible"

    # Test response headers
    http_response=$(curl -I -s "http://$GERRIT_HOST:$HTTP_PORT")
    if echo "$http_response" | grep -q "200 OK"; then
        echo "✅ HTTP returns 200 OK"
    else
        echo "⚠️  HTTP response: $(echo "$http_response" | head -1)"
    fi
else
    echo "❌ HTTP interface not accessible"
fi

# Test 3: HTTPS Connectivity (if configured)
echo -e "\n3️⃣  Testing HTTPS connectivity..."
if nc -zv $GERRIT_HOST $HTTPS_PORT 2>/dev/null; then
    echo "✅ HTTPS port accessible"
```

```
    if curl -s -f -k "https://$GERRIT_HOST:$HTTPS_PORT" > /dev/null; then
        echo "✅ HTTPS interface responding"

        # Test SSL certificate
        cert_info=$(echo | openssl s_client -connect $GERRIT_HOST:$HTTPS_PORT
2>/dev/null | openssl x509 -noout -dates 2>/dev/null)
        if [ $? -eq 0 ]; then
            echo "✅ SSL certificate valid"
            echo "$cert_info" | sed 's/^/    /'
        else
            echo "⚠️  SSL certificate issues detected"
        fi
    else
        echo "❌ HTTPS interface not responding"
    fi
else
    echo "ℹ️  HTTPS not configured or not accessible"
fi

# Test 4: SSH Connectivity
echo -e "\n4️⃣  Testing SSH connectivity..."
if nc -zv $GERRIT_HOST $SSH_PORT 2>/dev/null; then
    echo "✅ SSH port accessible"

    # Test SSH banner
    ssh_banner=$(echo | telnet $GERRIT_HOST $SSH_PORT 2>/dev/null | grep -i
gerrit)
    if [ -n "$ssh_banner" ]; then
        echo "✅ Gerrit SSH service responding"
    else
        echo "⚠️  SSH port open but may not be Gerrit service"
    fi
else
    echo "❌ SSH port not accessible"
fi

# Test 5: Configuration Files
echo -e "\n5️⃣  Testing configuration..."
if [ -f "$GERRIT_SITE/etc/gerrit.config" ]; then
    echo "✅ Main configuration file exists"

    # Check canonical URL
    canonical_url=$(grep -E "canonicalWebUrl" "$GERRIT_SITE/etc/gerrit.config"
| cut -d'=' -f2 | tr -d ' ')
    if [ -n "$canonical_url" ]; then
        echo "✅ Canonical URL configured: $canonical_url"
    else
        echo "⚠️  Canonical URL not configured"
    fi
else
    echo "❌ Configuration file not found"
fi
```

```
# Test 6: Database
echo -e "\n6️⃣  Testing database..."
if [ -d "$GERRIT_SITE/db" ]; then
    echo "✅ Database directory exists"

    db_files=$(ls -1 "$GERRIT_SITE/db" 2>/dev/null | wc -l)
    if [ $db_files -gt 0 ]; then
        echo "✅ Database files present ($db_files files)"
    else
        echo "⚠️  Database directory empty"
    fi
else
    echo "❌ Database directory not found"
fi

# Test 7: Git Repositories
echo -e "\n7️⃣  Testing Git repository storage..."
if [ -d "$GERRIT_SITE/git" ]; then
    echo "✅ Git repository directory exists"

    repo_count=$(find "$GERRIT_SITE/git" -name "*.git" -type d 2>/dev/null | wc
-l)
    echo "ℹ️  Number of Git repositories: $repo_count"
else
    echo "❌ Git repository directory not found"
fi

# Test 8: Logs
echo -e "\n8️⃣  Testing logs..."
if [ -d "$GERRIT_SITE/logs" ]; then
    echo "✅ Log directory exists"

    # Check for recent errors
    if [ -f "$GERRIT_SITE/logs/error_log" ]; then
        error_count=$(grep -i error "$GERRIT_SITE/logs/error_log" 2>/dev/null |
tail -10 | wc -l)
        if [ $error_count -eq 0 ]; then
            echo "✅ No recent errors in log"
        else
            echo "⚠️  Found $error_count recent errors (check error_log)"
        fi
    else
        echo "ℹ️  Error log not found (may be normal for new installation)"
    fi
else
    echo "❌ Log directory not found"
fi

# Test 9: Web Interface Features
echo -e "\n9️⃣  Testing web interface features..."
if command -v curl > /dev/null; then
    # Test admin interface
    admin_response=$(curl -s "http://$GERRIT_HOST:$HTTP_PORT/admin/projects" |
```

```
head -1)
    if echo "$admin_response" | grep -q "<!DOCTYPE html"; then
        echo "✅ Admin interface accessible"
    else
        echo "⚠️  Admin interface may not be accessible"
    fi

    # Test REST API
    api_response=$(curl -s
"http://$GERRIT_HOST:$HTTP_PORT/config/server/version" 2>/dev/null)
    if [ -n "$api_response" ]; then
        echo "✅ REST API responding"
        echo "ℹ️  Gerrit version: $api_response"
    else
        echo "⚠️  REST API not responding"
    fi
else
    echo "ℹ️  curl not available, skipping web interface tests"
fi

# Summary
echo -e "\n📋 Test Summary"
echo "=================="
echo "✅ = Passed"
echo "⚠️  = Warning/Attention needed"
echo "❌ = Failed"
echo "ℹ️  = Information only"
echo ""
echo "📄 If you see warnings or failures, check the respective sections"
echo "   in the installation guide for troubleshooting steps."
echo ""
echo "🎉 Gerrit installation test completed!"
```

Save and run the test script:

```
# Save the script
cat > test-gerrit-installation.sh << 'EOF'
[paste the script content above]
EOF

# Make executable
chmod +x test-gerrit-installation.sh

# Run the test
./test-gerrit-installation.sh localhost 8080 8443 29418
/home/gerrit/gerrit_site
```

Troubleshooting Test Failures

Common test failures and solutions:

1. **Service not running**

```
# Check why service failed to start
cat ~/gerrit/gerrit_site/logs/error_log

# Try starting manually
~/gerrit/gerrit_site/bin/gerrit.sh start
```

2. **HTTP not accessible**

```
# Check if port is in use
netstat -tulpn | grep :8080

# Check firewall
sudo ufw status
sudo firewall-cmd --list-all
```

3. **SSL/HTTPS issues**

```
# Check certificate files
ls -la ~/gerrit/gerrit_site/etc/ssl/

# Verify certificate validity
openssl x509 -in ~/gerrit/gerrit_site/etc/ssl/gerrit.crt -text -noout
```

4. **SSH not working**

```
# Check SSH daemon configuration
grep -i ssh ~/gerrit/gerrit_site/etc/gerrit.config

# Check if SSH keys are configured
ls -la ~/.ssh/
```

Next Steps

Congratulations! You now have a working Gerrit installation that has been thoroughly tested. In the next chapter, we'll explore the core concepts and terminology you need to understand to use Gerrit effectively.

Complete Verification Checklist

- ☐ Java is installed and working (`java -version`)

- [ ] Gerrit is downloaded and initialized
- [ ] Gerrit service is running (`./bin/gerrit.sh check`)
- [ ] HTTP interface accessible at http://localhost:8080
- [ ] HTTPS configured and working (if applicable)
- [ ] SSH daemon accessible on port 29418
- [ ] You can log in as admin user
- [ ] Test project can be created
- [ ] Configuration files are properly set up
- [ ] Database is initialized and accessible
- [ ] Git repository storage is working
- [ ] No critical errors in logs
- [ ] Performance tests pass

Ready to learn Gerrit concepts? Continue to Chapter 3: Understanding Gerrit Concepts

Summary

In this chapter, you learned:

- How to install Java and Gerrit from scratch
- How to initialize Gerrit with basic configuration
- How to configure HTTPS/SSL with Let's Encrypt and self-signed certificates
- How to start and manage the Gerrit service
- How to access and test the web interface
- How to create your first admin user
- How to perform comprehensive installation testing
- How to troubleshoot common installation issues

The foundation is now set for exploring Gerrit's powerful code review capabilities with a secure, well-tested installation!

Continue to Chapter 3: Understanding Gerrit Concepts