



Installation from source

- › Consider the Omnibus package installation
- › Select Version to Install
- › Important Notes
- › Overview
- › 1. Packages / Dependencies
- › 2. Ruby
- › 3. Go
- › 4. Node
- › 5. System Users
- › 6. Database
- › 7. Redis
- › 8. GitLab
 - › Clone the Source
 - › Configure It
 - › Configure GitLab DB Settings
 - › Install Gems
 - › Install GitLab Shell
 - › Install gitlab-workhorse
 - › Initialize Database and Activate Advanced Features
 - › Secure secrets.yml
 - › Install Init Script
 - › Install Gitaly
 - › Setup Logrotate
 - › Check Application Status
 - › Compile GetText PO files
 - › Compile Assets
 - › Start Your GitLab Instance

- › 9. Nginx
 - › Installation
 - › Site Configuration
 - › Test Configuration
 - › Restart
 - › Done!
 - › Double-check Application Status
 - › Initial Login
 - › Advanced Setup Tips
 - › Relative URL support
 - › Using HTTPS
 - › Enable Reply by email
 - › LDAP Authentication
 - › Using Custom Omniauth Providers
 - › Build your projects
 - › Adding your Trusted Proxies
 - › Custom Redis Connection
 - › Custom SSH Connection
 - › Additional Markup Styles
 - › Troubleshooting
 - › "You appear to have cloned an empty repository."
 - › google-protobuf "LoadError: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.14' not found"
-

[GitLab Documentation \(/ce/README.html\)](#) › [Installation \(/ce/install/README.html\)](#) › Installation from

Installation from source

Consider the Omnibus package installation

Since an installation from source is a lot of work and error prone we strongly recommend the fast and reliable Omnibus package installation (<https://about.gitlab.com/downloads/>) (deb/rpm).

One reason the Omnibus package is more reliable is its use of Runit to restart any of the GitLab processes in case one crashes. On heavily used GitLab instances the memory usage of the Sidekiq background worker will grow over time. Omnibus packages solve this by letting the Sidekiq terminate gracefully (http://docs.gitlab.com/ce/operations/sidekiq_memory_killer.html) if it uses too much memory. After this termination Runit will detect Sidekiq is not running and will start it. Since installations from source don't have Runit, Sidekiq can't be terminated and its memory usage will grow over time.

Select Version to Install

Make sure you view this installation guide (<https://gitlab.com/gitlab-org/gitlab-ce/blob/master/doc/install/installation.md>) from the tag (version) of GitLab you would like to install. In most cases this should be the highest numbered production tag (without rc in it). You can select the tag in the version dropdown in the top left corner of GitLab (below the menu bar).

If the highest number stable branch is unclear please check the GitLab Blog (<https://about.gitlab.com/blog/>) for installation guide links by version.

Important Notes

This guide is long because it covers many cases and includes all commands you need, this is one of the few installation scripts that actually works out of the box [↗](#).

This installation guide was created for and tested on **Debian/Ubuntu** operating systems. Please read [requirements.md](#) ([requirements.html](#)) for hardware and operating system requirements. If you want to install on RHEL/CentOS we recommend using the Omnibus packages (<https://about.gitlab.com/downloads/>).

This is the official installation guide to set up a production server. To set up a **development installation** or for many other installation options please see the installation section of the readme (<https://gitlab.com/gitlab-org/gitlab-ce/blob/master/README.md#installation>).

The following steps have been known to work. Please **use caution when you deviate** from this guide. Make sure you don't violate any assumptions GitLab makes about its environment. For example many people run into permission problems because they changed the location of directories or run services as the wrong user.

If you find a bug/error in this guide please **submit a merge request** following the contributing guide (<https://gitlab.com/gitlab-org/gitlab-ce/blob/master/CONTRIBUTING.md>).

Overview

The GitLab installation consists of setting up the following components:

1. Packages / Dependencies
2. Ruby
3. Go
4. Node

5. System Users
6. Database
7. Redis
8. GitLab
9. Nginx

1. Packages / Dependencies

`sudo` is not installed on Debian by default. Make sure your system is up-to-date and install it.

```
# run as root!  
apt-get update -y  
apt-get upgrade -y  
apt-get install sudo -y
```

Note: During this installation some files will need to be edited manually. If you are familiar with vim set it as default editor with the commands below. If you are not familiar with vim please skip this and keep using the default editor.

```
# Install vim and set as default editor  
sudo apt-get install -y vim  
sudo update-alternatives --set editor /usr/bin/vim.basic
```

Install the required packages (needed to compile Ruby and native extensions to Ruby gems):

```
sudo apt-get install -y build-essential zlib1g-dev libyaml-dev libssl-dev libgdb  
m-dev libre2-dev libreadline-dev libncurses5-dev libffi-dev curl openssh-server  
checkinstall libxml2-dev libxslt-dev libcurl4-openssl-dev libicu-dev logrotate p  
ython-docutils pkg-config cmake
```

Ubuntu 14.04 (Trusty Tahr) doesn't have the `libre2-dev` package available, but you can install `re2` manually [↗](#).

If you want to use Kerberos for user authentication, then install `libkrb5-dev`:

```
sudo apt-get install libkrb5-dev
```

Note: If you don't know what Kerberos is, you can assume you don't need it.

Make sure you have the right version of Git installed

```
# Install Git  
sudo apt-get install -y git-core  
  
# Make sure Git is version 2.13.0 or higher  
git --version
```

Is the system packaged Git too old? Remove it and compile from source.

```
# Remove packaged Git
sudo apt-get remove git-core

# Install dependencies
sudo apt-get install -y libcurl4-openssl-dev libexpat1-dev gettext libz-dev libs
sl-dev build-essential

# Download and compile from source
cd /tmp
curl --remote-name --progress https://www.kernel.org/pub/software/scm/git/git-2.
8.4.tar.gz
echo '626e319f8a24fc0866167ea5f6bf3e2f38f69d6cb2e59e150f13709ca3ebf301 git-2.8.
4.tar.gz' | shasum -a256 -c - && tar -xzf git-2.8.4.tar.gz
cd git-2.8.4/
./configure
make prefix=/usr/local all

# Install into /usr/local/bin
sudo make prefix=/usr/local install

# When editing config/gitlab.yml (Step 5), change the git -> bin_path to /usr/lo
cal/bin/git
```

Note: In order to receive mail notifications, make sure to install a mail server. By default, Debian is shipped with exim4 but this has problems [↗](#) while Ubuntu does not ship with one. The recommended mail server is postfix and you can install it with:

```
sudo apt-get install -y postfix
```

Then select 'Internet Site' and press enter to confirm the hostname.

2. Ruby

The Ruby interpreter is required to run GitLab.

Note: The current supported Ruby (MRI) version is 2.3.x. GitLab 9.0 dropped support for Ruby 2.1.x.

The use of Ruby version managers such as RVM [↗](#), rbenv [↗](#) or chruby [↗](#) with GitLab in production, frequently leads to hard to diagnose problems. For example, GitLab Shell is called from OpenSSH, and having a version manager can prevent pushing and pulling over SSH. Version managers are not supported and we strongly advise everyone to follow the instructions below to use a system Ruby.

Linux distributions generally have older versions of Ruby available, so these instructions are designed to install Ruby from the official source code.

Remove the old Ruby 1.8 if present:

```
sudo apt-get remove ruby1.8
```


Download Ruby and compile it:

```
mkdir /tmp/ruby && cd /tmp/ruby
curl --remote-name --progress https://cache.ruby-lang.org/pub/ruby/2.3/ruby-2.3.3.tar.gz
echo '1014ee699071aa2ddd501907d18cbe15399c997d ruby-2.3.3.tar.gz' | shasum -c - && tar xzf ruby-2.3.3.tar.gz
cd ruby-2.3.3
./configure --disable-install-rdoc
make
sudo make install
```

Then install the Bundler Gem:

```
sudo gem install bundler --no-ri --no-rdoc
```

3. Go

Since GitLab 8.0, GitLab has several daemons written in Go. To install GitLab we need a Go compiler. The instructions below assume you use 64-bit Linux. You can find downloads for other platforms at the [Go download page](#) .

```
# Remove former Go installation folder
sudo rm -rf /usr/local/go

curl --remote-name --progress https://storage.googleapis.com/golang/go1.8.3.linux-amd64.tar.gz
echo '1862f4c3d3907e59b04a757cfda0ea7aa9ef39274af99a784f5be843c80c6772 go1.8.3.linux-amd64.tar.gz' | shasum -a256 -c - && \
  sudo tar -C /usr/local -xzf go1.8.3.linux-amd64.tar.gz
sudo ln -sf /usr/local/go/bin/{go,godoc,gofmt} /usr/local/bin/
rm go1.8.3.linux-amd64.tar.gz
```

4. Node

Since GitLab 8.17, GitLab requires the use of node \geq v4.3.0 to compile javascript assets, and yarn \geq v0.17.0 to manage javascript dependencies. In many distros the versions provided by the official package repositories are out of date, so we'll need to install through the following commands:

```
# install node v7.x
curl --location https://deb.nodesource.com/setup_7.x | sudo bash -
sudo apt-get install -y nodejs

curl --silent --show-error https://dl.yarnpkg.com/debian/pubkey.gpg | sudo apt-key add -
echo "deb https://dl.yarnpkg.com/debian/ stable main" | sudo tee /etc/apt/sources.list.d/yarn.list
sudo apt-get update
sudo apt-get install yarn
```

Visit the official websites for [node](#) and [yarn](#) if you have any trouble with these steps.

5. System Users

Create a `git` user for GitLab:

```
sudo adduser --disabled-login --gecos 'GitLab' git
```

6. Database

We recommend using a PostgreSQL database. For MySQL check the MySQL setup guide ([database_mysql.html](#)).

Note: because we need to make use of extensions and concurrent index removal, you need at least PostgreSQL 9.2.

1. Install the database packages:

```
sudo apt-get install -y postgresql postgresql-client libpq-dev postgresql-contrib
```

2. Create a database user for GitLab:

```
sudo -u postgres psql -d template1 -c "CREATE USER git CREATEDB;"
```

3. Create the `pg_trgm` extension (required for GitLab 8.6+):

```
sudo -u postgres psql -d template1 -c "CREATE EXTENSION IF NOT EXISTS pg_trgm;"
```

4. Create the GitLab production database and grant all privileges on database:

```
sudo -u postgres psql -d template1 -c "CREATE DATABASE gitlabhq_production OWNER git;"
```

5. Try connecting to the new database with the new user:

```
sudo -u git -H psql -d gitlabhq_production
```

6. Check if the `pg_trgm` extension is enabled:

```
SELECT true AS enabled
FROM pg_available_extensions
WHERE name = 'pg_trgm'
AND installed_version IS NOT NULL;
```

If the extension is enabled this will produce the following output:

```
enabled
-----
t
(1 row)
```

7. Quit the database session:

```
gitlabhq_production> \q
```

7. Redis

GitLab requires at least Redis 2.8.

If you are using Debian 8 or Ubuntu 14.04 and up, then you can simply install Redis 2.8 with:

```
sudo apt-get install redis-server
```

If you are using Debian 7 or Ubuntu 12.04, follow the special documentation on an alternate Redis installation ([redis.html](#)). Once done, follow the rest of the guide here.


```
# Configure redis to use sockets
sudo cp /etc/redis/redis.conf /etc/redis/redis.conf.orig

# Disable Redis listening on TCP by setting 'port' to 0
sed 's/^port .*/port 0/' /etc/redis/redis.conf.orig | sudo tee /etc/redis/redis.conf

# Enable Redis socket for default Debian / Ubuntu path
echo 'unixsocket /var/run/redis/redis.sock' | sudo tee -a /etc/redis/redis.conf

# Grant permission to the socket to all members of the redis group
echo 'unixsocketperm 770' | sudo tee -a /etc/redis/redis.conf

# Create the directory which contains the socket
mkdir /var/run/redis
chown redis:redis /var/run/redis
chmod 755 /var/run/redis

# Persist the directory which contains the socket, if applicable
if [ -d /etc/tmpfiles.d ]; then
    echo 'd /var/run/redis 0755 redis redis 10d -' | sudo tee -a /etc/tmpfiles.d/redis.conf
fi

# Activate the changes to redis.conf
sudo service redis-server restart

# Add git to the redis group
sudo usermod -aG redis git
```

8. GitLab

```
# We'll install GitLab into home directory of the user "git"
cd /home/git
```

Clone the Source

```
# Clone GitLab repository
sudo -u git -H git clone https://gitlab.com/gitlab-org/gitlab-ce.git -b 9-5-stable gitlab
```

Note: You can change `9-5-stable` to `master` if you want the *bleeding edge* version, but never install `master` on a production server!

Configure It

```
# Go to GitLab installation folder
cd /home/git/gitlab

# Copy the example GitLab config
sudo -u git -H cp config/gitlab.yml.example config/gitlab.yml

# Update GitLab config file, follow the directions at top of file
sudo -u git -H editor config/gitlab.yml

# Copy the example secrets file
sudo -u git -H cp config/secrets.yml.example config/secrets.yml
sudo -u git -H chmod 0600 config/secrets.yml

# Make sure GitLab can write to the log/ and tmp/ directories
sudo chown -R git log/
sudo chown -R git tmp/
sudo chmod -R u+rwx,go-w log/
sudo chmod -R u+rwx tmp/

# Make sure GitLab can write to the tmp/pids/ and tmp/sockets/ directories
sudo chmod -R u+rwx tmp/pids/
sudo chmod -R u+rwx tmp/sockets/

# Create the public/uploads/ directory
sudo -u git -H mkdir public/uploads/

# Make sure only the GitLab user has access to the public/uploads/ directory
# now that files in public/uploads are served by gitlab-workhorse
sudo chmod 0700 public/uploads

# Change the permissions of the directory where CI job traces are stored
sudo chmod -R u+rwx builds/

# Change the permissions of the directory where CI artifacts are stored
sudo chmod -R u+rwx shared/artifacts/

# Change the permissions of the directory where GitLab Pages are stored
sudo chmod -R ug+rwx shared/pages/

# Copy the example Unicorn config
sudo -u git -H cp config/unicorn.rb.example config/unicorn.rb

# Find number of cores
nproc

# Enable cluster mode if you expect to have a high load instance
# Set the number of workers to at least the number of cores
# Ex. change amount of workers to 3 for 2GB RAM server
sudo -u git -H editor config/unicorn.rb

# Copy the example Rack attack config
sudo -u git -H cp config/initializers/rack_attack.rb.example config/initializers/
/rack_attack.rb

# Configure Git global settings for git user
# 'autocrlf' is needed for the web editor
sudo -u git -H git config --global core.autocrlf input
```

```
# Disable 'git gc --auto' because GitLab already runs 'git gc' when needed
sudo -u git -H git config --global gc.auto 0

# Enable packfile bitmaps
sudo -u git -H git config --global repack.writeBitmaps true

# Configure Redis connection settings
sudo -u git -H cp config/resque.yml.example config/resque.yml

# Change the Redis socket path if you are not using the default Debian / Ubuntu
configuration
sudo -u git -H editor config/resque.yml
```

Important Note: Make sure to edit both `gitlab.yml` and `unicorn.rb` to match your setup.

Note: If you want to use HTTPS, see [Using HTTPS](#) for the additional steps.

Configure GitLab DB Settings

```
# PostgreSQL only:
sudo -u git cp config/database.yml.postgresql config/database.yml

# MySQL only:
sudo -u git cp config/database.yml.mysql config/database.yml

# MySQL and remote PostgreSQL only:
# Update username/password in config/database.yml.
# You only need to adapt the production settings (first part).
# If you followed the database guide then please do as follows:
# Change 'secure password' with the value you have given to $password
# You can keep the double quotes around the password
sudo -u git -H editor config/database.yml

# PostgreSQL and MySQL:
# Make config/database.yml readable to git only
sudo -u git -H chmod o-rwx config/database.yml
```

Install Gems

Note: As of bundler 1.5.2, you can invoke `bundle install -jN` (where N the number of your processor cores) and enjoy the parallel gems installation with measurable difference in completion time (~60% faster). Check the number of your cores with `nproc`. For more information check this [post](#). First make sure you have bundler >= 1.5.2 (run `bundle -v`) as it addresses some issues that were fixed in 1.5.2.

```
# For PostgreSQL (note, the option says "without ... mysql")
sudo -u git -H bundle install --deployment --without development test mysql aws
kerberos

# Or if you use MySQL (note, the option says "without ... postgres")
sudo -u git -H bundle install --deployment --without development test postgres a
ws kerberos
```

Note: If you want to use Kerberos for user authentication, then omit `kerberos` in the `--without` option above.

Install GitLab Shell

GitLab Shell is an SSH access and repository management software developed specially for GitLab.

```
# Run the installation task for gitlab-shell (replace `REDIS_URL` if needed):
sudo -u git -H bundle exec rake gitlab:shell:install REDIS_URL=unix:/var/run/red
is/redis.sock RAILS_ENV=production SKIP_STORAGE_VALIDATION=true

# By default, the gitlab-shell config is generated from your main GitLab config.
# You can review (and modify) the gitlab-shell config as follows:
sudo -u git -H editor /home/git/gitlab-shell/config.yml
```

Note: If you want to use HTTPS, see [Using HTTPS](#) for the additional steps.

Note: Make sure your hostname can be resolved on the machine itself by either a proper DNS record or an additional line in `/etc/hosts` ("`127.0.0.1 hostname`"). This might be necessary for example if you set up GitLab behind a reverse proxy. If the hostname cannot be resolved, the final installation check will fail with "Check GitLab API access: FAILED. code: 401" and pushing commits will be rejected with "[remote rejected] master -> master (hook declined)".

Note: GitLab Shell application startup time can be greatly reduced by disabling RubyGems. This can be done in several manners:

- Export `RUBYOPT=--disable-gems` environment variable for the processes
- Compile Ruby with `configure --disable-rubygems` to disable RubyGems by default. Not recommended for system-wide Ruby.
- Omnibus GitLab replaces the *shebang* line of the `gitlab-shell/bin/*` scripts (https://gitlab.com/gitlab-org/omnibus-gitlab/merge_requests/1707)

Install gitlab-workhorse

GitLab-Workhorse uses GNU Make [↗](#). The following command-line will install GitLab-Workhorse in `/home/git/gitlab-workhorse` which is the recommended location.

```
sudo -u git -H bundle exec rake "gitlab:workhorse:install[/home/git/gitlab-workh
orserse]" RAILS_ENV=production
```

You can specify a different Git repository by providing it as an extra paramter:

```
sudo -u git -H bundle exec rake "gitlab:workhorse:install[/home/git/gitlab-workhorse,https://example.com/gitlab-workhorse.git]" RAILS_ENV=production
```

Initialize Database and Activate Advanced Features

```
sudo -u git -H bundle exec rake gitlab:setup RAILS_ENV=production

# Type 'yes' to create the database tables.

# When done you see 'Administrator account created:'
```

Note: You can set the Administrator/root password and e-mail by supplying them in environmental variables, `GITLAB_ROOT_PASSWORD` and `GITLAB_ROOT_EMAIL` respectively, as seen below. If you don't set the password (and it is set to the default one) please wait with exposing GitLab to the public internet until the installation is done and you've logged into the server the first time. During the first login you'll be forced to change the default password.

```
sudo -u git -H bundle exec rake gitlab:setup RAILS_ENV=production GITLAB_ROOT_PASSWORD=yourpassword GITLAB_ROOT_EMAIL=youremail
```

Secure secrets.yml

The `secrets.yml` file stores encryption keys for sessions and secure variables. Backup `secrets.yml` someplace safe, but don't store it in the same place as your database backups. Otherwise your secrets are exposed if one of your backups is compromised.

Install Init Script

Download the init script (will be `/etc/init.d/gitlab`):

```
sudo cp lib/support/init.d/gitlab /etc/init.d/gitlab
```

And if you are installing with a non-default folder or user copy and edit the defaults file:

```
sudo cp lib/support/init.d/gitlab.default.example /etc/default/gitlab
```

If you installed GitLab in another directory or as a user other than the default you should change these settings in `/etc/default/gitlab`. Do not edit `/etc/init.d/gitlab` as it will be changed on upgrade.

Make GitLab start on boot:

```
sudo update-rc.d gitlab defaults 21
```

Install Gitaly

```
# Fetch Gitaly source with Git and compile with Go
sudo -u git -H bundle exec rake "gitlab:gitaly:install[/home/git/gitaly]" RAILS_
ENV=production
```

You can specify a different Git repository by providing it as an extra paramter:

```
sudo -u git -H bundle exec rake "gitlab:gitaly:install[/home/git/gitaly,https://
example.com/gitaly.git]" RAILS_ENV=production
```

Next, make sure gitaly configured:

```
# Restrict Gitaly socket access
sudo chmod 0700 /home/git/gitlab/tmp/sockets/private
sudo chown git /home/git/gitlab/tmp/sockets/private

# If you are using non-default settings you need to update config.toml
cd /home/git/gitaly
sudo -u git -H editor config.toml
```

For more information about configuring Gitaly see [doc/administration/gitaly](#) ([../administration/gitaly](#)).

Setup Logrotate

```
sudo cp lib/support/logrotate/gitlab /etc/logrotate.d/gitlab
```

Check Application Status

Check if GitLab and its environment are configured correctly:

```
sudo -u git -H bundle exec rake gitlab:env:info RAILS_ENV=production
```

Compile GetText PO files

```
sudo -u git -H bundle exec rake gettext:pack RAILS_ENV=production
sudo -u git -H bundle exec rake gettext:po_to_json RAILS_ENV=production
```

Compile Assets

```
sudo -u git -H yarn install --production --pure-lockfile
sudo -u git -H bundle exec rake gitlab:assets:compile RAILS_ENV=production NODE_
ENV=production
```

Start Your GitLab Instance

```
sudo service gitlab start
# or
sudo /etc/init.d/gitlab restart
```

9. Nginx

Note: Nginx is the officially supported web server for GitLab. If you cannot or do not want to use Nginx as your web server, have a look at the GitLab recipes (<https://gitlab.com/gitlab-org/gitlab-recipes/>).

Installation

```
sudo apt-get install -y nginx
```

Site Configuration

Copy the example site config:

```
sudo cp lib/support/nginx/gitlab /etc/nginx/sites-available/gitlab
sudo ln -s /etc/nginx/sites-available/gitlab /etc/nginx/sites-enabled/gitlab
```

Make sure to edit the config file to match your setup. Also, ensure that you match your paths to GitLab, especially if installing for a user other than the 'git' user:

```
# Change YOUR_SERVER_FQDN to the fully-qualified
# domain name of your host serving GitLab.
#
# Remember to match your paths to GitLab, especially
# if installing for a user other than 'git'.
#
# If using Ubuntu default nginx install:
# either remove the default_server from the listen line
# or else sudo rm -f /etc/nginx/sites-enabled/default
sudo editor /etc/nginx/sites-available/gitlab
```

If you intend to enable GitLab pages, there is a separate Nginx config you need to use. Read all about the needed configuration at the GitLab Pages administration guide ([../administration/pages/index.html](https://docs.gitlab.com/ce/administration/pages/index.html)).

Note: If you want to use HTTPS, replace the `gitlab` Nginx config with `gitlab-ssl`. See Using

HTTPS for HTTPS configuration details.

Test Configuration

Validate your `gitlab` or `gitlab-ssl` Nginx config file with the following command:

```
sudo nginx -t
```

You should receive `syntax is okay` and `test is successful` messages. If you receive errors check your `gitlab` or `gitlab-ssl` Nginx config file for typos, etc. as indicated in the error message given.

Restart

```
sudo service nginx restart
```

Done!

Double-check Application Status

To make sure you didn't miss anything run a more thorough check with:

```
sudo -u git -H bundle exec rake gitlab:check RAILS_ENV=production
```

If all items are green, then congratulations on successfully installing GitLab!

NOTE: Supply `SANITIZE=true` environment variable to `gitlab:check` to omit project names from the output of the check command.

Initial Login

Visit `YOUR_SERVER` in your web browser for your first GitLab login.

If you didn't provide a root password during setup, you'll be redirected to a password reset screen to provide the password for the initial administrator account. Enter your desired password and you'll be redirected back to the login screen.

The default account's username is **root**. Provide the password you created earlier and login. After login you can change the username if you wish.

Enjoy!

You can use `sudo service gitlab start` and `sudo service gitlab stop` to start and stop GitLab.

Advanced Setup Tips

Relative URL support

See the Relative URL documentation ([relative_url.html](#)) for more information on how to configure GitLab with a relative URL.

Using HTTPS

To use GitLab with HTTPS:

1. In `gitlab.yml`:
 1. Set the `port` option in section 1 to `443`.
 2. Set the `https` option in section 1 to `true`.
2. In the `config.yml` of `gitlab-shell`:
 1. Set `gitlab_url` option to the HTTPS endpoint of GitLab (e.g. `https://git.example.com`).
 2. Set the certificates using either the `ca_file` or `ca_path` option.
3. Use the `gitlab-ssl` Nginx example config instead of the `gitlab` config.
 1. Update `YOUR_SERVER_FQDN`.
 2. Update `ssl_certificate` and `ssl_certificate_key`.
 3. Review the configuration file and consider applying other security and performance enhancing features.

Using a self-signed certificate is discouraged but if you must use it follow the normal directions then:

1. Generate a self-signed SSL certificate:

```
mkdir -p /etc/nginx/ssl/  
cd /etc/nginx/ssl/  
sudo openssl req -newkey rsa:2048 -x509 -nodes -days 3560 -out gitlab.crt  
-keyout gitlab.key  
sudo chmod o-r gitlab.key
```

2. In the `config.yml` of `gitlab-shell` set `self_signed_cert` to `true`.

Enable Reply by email

See the "Reply by email" documentation ([../administration/reply_by_email.html](#)) for more information on how to set this up.

LDAP Authentication

You can configure LDAP authentication in `config/gitlab.yml`. Please restart GitLab after editing

this file.

Using Custom Omniauth Providers

See the omniauth integration document ([../integration/omniauth.html](https://docs.gitlab.com/ce/integration/omniauth.html))

Build your projects

GitLab can build your projects. To enable that feature you need GitLab Runners to do that for you. Checkout the GitLab Runner section (<https://about.gitlab.com/gitlab-ci/#gitlab-runner>) to install it

Adding your Trusted Proxies

If you are using a reverse proxy on an separate machine, you may want to add the proxy to the trusted proxies list. Otherwise users will appear signed in from the proxy's IP address.

You can add trusted proxies in `config/gitlab.yml` by customizing the `trusted_proxies` option in section 1. Save the file and reconfigure GitLab ([../administration/restart_gitlab.html](https://docs.gitlab.com/ce/administration/restart_gitlab.html)) for the changes to take effect.

Custom Redis Connection

If you'd like to connect to a Redis server on a non-standard port or on a different host, you can configure its connection string via the `config/resque.yml` file.

```
# example
production:
  url: redis://redis.example.tld:6379
```

If you want to connect the Redis server via socket, then use the "unix:" URL scheme and the path to the Redis socket file in the `config/resque.yml` file.

```
# example
production:
  url: unix:/path/to/redis/socket
```

Also you can use environment variables in the `config/resque.yml` file:

```
# example
production:
  url: <%= ENV.fetch('GITLAB_REDIS_URL') %>
```


Custom SSH Connection

If you are running SSH on a non-standard port, you must change the GitLab user's SSH config.

```
# Add to /home/git/.ssh/config
host localhost          # Give your setup a name (here: override localhost)
  user git              # Your remote git user
  port 2222            # Your port number
  hostname 127.0.0.1; # Your server name or IP
```

You also need to change the corresponding options (e.g. `ssh_user` , `ssh_host` , `admin_uri`) in the `config\gitlab.yml` file.

Additional Markup Styles

Apart from the always supported markdown style there are other rich text files that GitLab can display. But you might have to install a dependency to do so. Please see the [github-markup gem readme](#)  for more information.

Troubleshooting

"You appear to have cloned an empty repository."

If you see this message when attempting to clone a repository hosted by GitLab, this is likely due to an outdated Nginx or Apache configuration, or a missing or misconfigured gitlab-workhorse instance. Double-check that you've installed Go, installed gitlab-workhorse, and correctly configured Nginx.

google-protobuf "LoadError: /lib/x86_64-linux-gnu/libc.so.6: version `GLIBC_2.14' not found"

This can happen on some platforms for some versions of the google-protobuf gem. The workaround is to install a source-only version of this gem ([google-protobuf.html](#)).

 [Edit this page \(https://gitlab.com/gitlab-org/gitlab-ce/blob/master/doc/install/installation.md\)](https://gitlab.com/gitlab-org/gitlab-ce/blob/master/doc/install/installation.md)



Products

Features

Enterprise Edition

Community Edition

Services

Resellers

Consultancy

Development

Created with Nanoc, hosted on GitLab Pages

[GitLab.com](#)

[Pricing](#)

Company

[Source Code](#)

[Blog](#)

[Press and Logos](#)

[About Us](#)

[Team](#)

[Direction](#)

[Handbook](#)

[Open Jobs](#)

[Terms](#)

[Contact Us](#)

[Training](#)

Community

[Events](#)

[Core Team](#)

[Contributors](#)

[Documentation](#)

[Getting Help](#)

[Contributing](#)

[Applications](#)

[Hall of Fame](#)