GitLab and SSH keys

Git is a distributed version control system, which means you can work locally but you can also share or "push" your changes to other servers. Before you can push your changes to a GitLab server you need a secure communication channel for sharing information.

The SSH protocol provides this security and allows you to authenticate to the GitLab remote server without supplying your username or password each time.

For a more detailed explanation of how the SSH protocol works, read this nice tutorial by DigitalOcean.

Requirements

The only requirement is to have the OpenSSH client installed on your system. This comes pre-installed on GNU/Linux and macOS, but not on Windows.

Depending on your Windows version, there are different methods to work with SSH keys.

Windows 10: Windows Subsystem for Linux

Starting with Windows 10, you can install the Windows Subsystem for Linux (WSL) where you can run Linux distributions directly on Windows, without the overhead of a virtual machine. Once installed and set up, you'll have the Git and SSH clients at your disposal.

Windows 10, 8.1, and 7: Git for Windows

The easiest way to install Git and the SSH client on Windows 8.1 and Windows 7 is Git for Windows. It provides a Bash emulation (Git Bash) used for running Git from the command line and the `ssh-keygen` command that is useful to create SSH keys as you'll learn below.

NOTE: **Alternative tools:** Although not explored in this page, you can use some alternative tools. Cygwin is a large collection of GNU and open source tools which provide functionality similar to a Unix distribution. PuttyGen provides a graphical user interface to create SSH keys.

Types of SSH keys and which to choose

GitLab supports RSA, DSA, ECDSA, and ED25519 keys. Their difference lies on the signing algorithm, and some of them have advantages over the others. For more information, you can read this nice article on ArchWiki. We'll focus on ED25519 and RSA and here.

NOTE: **Note:** As an admin, you can restrict which keys should be permitted and their minimum length. By default, all keys are permitted, which is also the case for GitLab.com.

ED25519 SSH keys

Following best practices, you should always favor ED25519 SSH keys, since they are more secure and have better performance over the other types.

ED25519 SSH keys were introduced in OpenSSH 6.5, so any modern OS should include the option to create them. If for any reason your OS or the GitLab instance you interact with doesn't support ED25519, you can fallback to RSA.

NOTE: **Note:** Omnibus does not ship with OpenSSH, so it uses the version on your GitLab server. If using Omnibus, ensure the version of OpenSSH installed is version 6.5 or newer if you want to use ED25519 SSH keys.

RSA SSH keys

RSA keys are the most common ones and therefore the most compatible with servers that may have an old OpenSSH version. Use them if the GitLab server doesn't work with ED25519 keys.

The minimum key size is 1024 bits, defaulting to 2048. If you wish to generate a stronger RSA key pair, specify the `-b` flag with a higher bit value than the default.

The old, default password encoding for SSH private keys is insecure; it's only a single round of an MD5 hash. Since OpenSSH version 6.5, you should use the `-o` option to `ssh-keygen` to encode your private key in a new, more secure format.

If you already have an RSA SSH key pair to use with GitLab, consider upgrading it to use the more secure password encryption format by using the following command on the private key:

```
ssh-keygen -o -f ~/.ssh/id_rsa
```

Generating a new SSH key pair

Before creating an SSH key pair, make sure to understand the different types of keys.

To create a new SSH key pair:

1. Open a terminal on Linux or macOS, or Git Bash / WSL on Windows.

2. Generate a new ED25519 SSH key pair:

   ```
   ssh-keygen -t ed25519 -C "email@example.com"
   ```

   Or, if you want to use RSA:

   ```
   ssh-keygen -o -t rsa -b 4096 -C "email@example.com"
   ```

   The `-C` flag adds a comment in the key in case you have multiple of them and want to tell which is which. It is optional.

3. Next, you will be prompted to input a file path to save your SSH key pair to. If you don't already have an SSH key pair and aren't generating a deploy key, use the suggested path by pressing `Enter`. Using the suggested path will normally allow your SSH client to automatically use the SSH key pair with no additional configuration.

   If you already have an SSH key pair with the suggested file path, you will need to input a new file path and declare what host this SSH key pair will be used for in your `~/.ssh/config` file.

4. Once the path is decided, you will be prompted to input a password to secure your new SSH key pair. It's a best practice to use a password, but it's not required and you can skip creating it by pressing `Enter` twice.

   If, in any case, you want to add or change the password of your SSH key pair, you can use the `-p` flag:

   ```
   ssh-keygen -p -o -f <keyname>
   ```

Now, it's time to add the newly created public key to your GitLab account.

Adding an SSH key to your GitLab account

1. Copy your **public** SSH key to the clipboard by using one of the commands below depending on your Operating System:

   macOS:

   ```
   pbcopy < ~/.ssh/id_ed25519.pub
   ```

   WSL / GNU/Linux (requires the xclip package):

   ```
   xclip -sel clip < ~/.ssh/id_ed25519.pub
   ```

   Git Bash on Windows:

   ```
   cat ~/.ssh/id_ed25519.pub | clip
   ```

   You can also open the key in a graphical editor and copy it from there, but be careful not to accidentally change anything.

   NOTE: **Note:** If you opted to create an RSA key, the name might differ.

2. Add your **public** SSH key to your GitLab account by:
   1. Clicking your avatar in the upper right corner and selecting **Settings**.
   2. Navigating to **SSH Keys** and pasting your **public** key in the **Key** field. If you:
      - Created the key with a comment, this will appear in the **Title** field.
      - Created the key without a comment, give your key an identifiable title like *Work Laptop* or *Home Workstation*.
   3. Click the **Add key** button.

   NOTE: **Note:** If you manually copied your public SSH key make sure you copied the entire key starting with `ssh-ed25519` (or `ssh-rsa`) and ending with your email.

Testing that everything is set up correctly

To test whether your SSH key was added correctly, run the following command in your terminal (replacing `gitlab.com` with your GitLab's instance domain):

```
ssh -T git@gitlab.com
```

The first time you connect to GitLab via SSH, you will be asked to verify the authenticity of the GitLab host you are connecting to. For example, when connecting to GitLab.com, answer `yes` to add GitLab.com to the list of trusted hosts:

```
The authenticity of host 'gitlab.com (35.231.145.151)' can't be established.
ECDSA key fingerprint is SHA256:HbW3g8zUjNSksFbqTiUWPWg2Bq1x8xdGUrliXFzSnUw.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'gitlab.com' (ECDSA) to the list of known hosts.
```

NOTE: **Note:** For GitLab.com, consult the SSH host keys fingerprints, to make sure you're connecting to the correct server.

Once added to the list of known hosts, you won't be asked to validate the authenticity of GitLab's host again. Run the above command once more, and you should only receive a *Welcome to GitLab, `@username`!* message.

If the welcome message doesn't appear, run SSH's verbose mode by replacing `-T` with `-vvvT` to understand where the error is.

Working with non-default SSH key pair paths

If you used a non-default file path for your GitLab SSH key pair, you must configure your SSH client to find your GitLab private SSH key for connections to GitLab.

Open a terminal and use the following commands (replacing `other_id_rsa` with your private SSH key):

```
eval $(ssh-agent -s)
ssh-add ~/.ssh/other_id_rsa
```

To retain these settings, you'll need to save them to a configuration file. For OpenSSH clients this is configured in the `~/.ssh/config` file. In this file you can set up configurations for multiple hosts, like GitLab.com, your own GitLab instance, GitHub, Bitbucket, etc.

Below are two example host configurations using their own SSH key:

```
# GitLab.com
Host gitlab.com
  Preferredauthentications publickey
  IdentityFile ~/.ssh/gitlab_com_rsa

# Private GitLab instance
Host gitlab.company.com
  Preferredauthentications publickey
  IdentityFile ~/.ssh/example_com_rsa
```

Public SSH keys need to be unique to GitLab, as they will bind to your account. Your SSH key is the only identifier you'll have when pushing code via SSH, that's why it needs to uniquely map to a single user.

Per-repository SSH keys

If you want to use different keys depending on the repository you are working on, you can issue the following command while inside your repository:

```
git config core.sshCommand "ssh -o IdentitiesOnly=yes -i ~/.ssh/private-key-filename-for-this-repository -F /dev/null"
```

This will not use the SSH Agent and requires at least Git 2.10.

Deploy keys

Per-repository deploy keys

Deploy keys allow read-only or read-write (if enabled) access to one or multiple projects with a single SSH key pair.

This is really useful for cloning repositories to your Continuous Integration (CI) server. By using deploy keys, you don't have to set up a dummy user account.

If you are a project maintainer or owner, you can add a deploy key in the project's **Settings > Repository** page by expanding the **Deploy Keys** section. Specify a title for the new deploy key and paste a public SSH key. After this, the machine that uses the corresponding private SSH key has read-only or read-write (if enabled) access to the project.

You can't add the same deploy key twice using the form. If you want to add the same key to another project, please enable it in the list that says 'Deploy keys from projects available to you'. All the deploy keys of all the projects you have access to are available. This project access can happen through being a direct member of the project, or through a group.

Deploy keys can be shared between projects, you just need to add them to each project.

Global shared deploy keys

Global Shared Deploy keys allow read-only or read-write (if enabled) access to be configured on any repository in the entire GitLab installation.

This is really useful for integrating repositories to secured, shared Continuous Integration (CI) services or other shared services. GitLab administrators can set up the Global Shared Deploy key in GitLab and add the private key to any shared systems. Individual repositories opt into exposing their repository using these keys when a project maintainers (or higher) authorizes a Global Shared Deploy key to be used with their project.

Global Shared Keys can provide greater security compared to Per-Project Deploy Keys since an administrator of the target integrated system is the only one who needs to know and configure the private key.

GitLab administrators set up Global Deploy keys in the Admin area under the section **Deploy Keys**. Ensure keys have a meaningful title as that will be the primary way for project maintainers and owners to identify the correct Global Deploy key to add. For instance, if the key gives access to a SaaS CI instance, use the name of that service in the key name if that is all it is used for. When creating Global Shared Deploy keys, give some thought to the granularity of keys - they could be of very narrow usage such as just a specific service or of broader usage for something like "Anywhere you need to give read access to your repository".

Once a GitLab administrator adds the Global Deployment key, project maintainers and owners can add it in project's **Settings > Repository** page by expanding the **Deploy Keys** section and clicking **Enable** next to the appropriate key listed under **Public deploy keys available to any project**.

NOTE: **Note:** The heading **Public deploy keys available to any project** only appears if there is at least one Global Deploy Key configured.

CAUTION: **Warning:** Defining Global Deploy Keys does not expose any given repository via the key until that repository adds the Global Deploy Key to their project. In this way the Global Deploy Keys enable access by other systems, but do not implicitly give any access just by setting them up.

Applications

Eclipse

If you are using EGit, you can add your SSH key to Eclipse.

SSH on the GitLab server

GitLab integrates with the system-installed SSH daemon, designating a user (typically named `git`) through which all access requests are handled. Users connecting to the GitLab server over SSH are identified by their SSH key instead of their username.

SSH *client* operations performed on the GitLab server wil be executed as this user. Although it is possible to modify the SSH configuration for this user to, e.g., provide a private SSH key to authenticate these requests by, this practice is **not supported** and is strongly discouraged as it presents significant security risks.

The GitLab check process includes a check for this condition, and will direct you to this section if your server is configured like this, e.g.:

```
$ gitlab-rake gitlab:check
# ...
Git user has default SSH configuration? ... no
  Try fixing it:
  mkdir ~/gitlab-check-backup-1504540051
  sudo mv /var/lib/git/.ssh/id_rsa ~/gitlab-check-backup-1504540051
  sudo mv /var/lib/git/.ssh/id_rsa.pub ~/gitlab-check-backup-1504540051
  For more information see:
  doc/ssh/README.md in section "SSH on the GitLab server"
  Please fix the error above and rerun the checks.
```

Remove the custom configuration as soon as you're able to. These customizations are *explicitly not supported* and may stop working at any time.

Troubleshooting

If on Git clone you are prompted for a password like `git@gitlab.com's password:` something is wrong with your SSH setup.

- Ensure that you generated your SSH key pair correctly and added the public SSH key to your GitLab profile
- Try manually registering your private SSH key using `ssh-agent` as documented earlier in this document
- Try to debug the connection by running `ssh -Tv git@example.com` (replacing `example.com` with your GitLab domain)