# TeamForge Git integration - Gerrit 2.1.x (for TeamForge 7.1)

# Contents

**TeamForge Git integration....4**
**Set up the TeamForge Git integration....6**
Requirements for the TeamForge Git integration....6
Install the TeamForge Git integration....6
Upgrade the Git integration along with TeamForge....7
Upgrade the Git integration independent of TeamForge....8
Reconfigure the TeamForge Git integration....9
Uninstall the TeamForge Git integration....9
TeamForge Git integration: VMware image....9
Change the administrator passwords (VMware image)....9
Change the administrator's SSH key pair (VMware image)....10
**Create a Git repository in TeamForge....12**
**Control access to Git source code....14**
**Access Git with an SSH key....15**
**Clone a Git repository....16**
**Work with Gerrit....17**
Add Gerrit as a linked application....17
Control the code review policy for Git repositories....17
Mandatory code reviews for Git repositories....18
Optional code review for Git repositories....19
Default code review for Git repositories....20
Custom code review for Git repositories....21
Code review policies: mappings....23
Notes on customizing your code review policy....23
Set up a code review process for TeamForge Git repositories....25
Prepare the Git client for code review....25
TeamForge Git integration: Gerrit Code Review policies FAQ....26
TeamForge Git integration: Gerrit Code Review workflow FAQ....27
Add a TeamForge user to Gerrit Administrators....27
Update Git repository access rights in Gerrit....29
Manage Gerrit access rights outside TeamForge....30
Set up Code Search for the TeamForge Git integration....32
Gerrit configuration (for advanced users)....34
Memory settings in Gerrit configuration....34
sshd settings in Gerrit configuration....35
Database settings in Gerrit configuration....36
log4j settings in Gerrit configuration....37
**TeamForge Git integration: History protection....38**
TeamForge Git integration: Enable history protection....38

History protection reports ....40
TeamForge Git integration: History protection FAQ....41
GERRIT_FORCE_HISTORY_PROTECTION....42
**TeamForge Git integration: FAQ....44**
TeamForge Git integration: Install FAQ....44
TeamForge Git integration: Post-install FAQ....44
TeamForge Git integration: General usage FAQ....46
TeamForge Git integration: Upgrade and Uninstall FAQ....49
TeamForge Git integration: Technical concepts FAQ....49
**TeamForge Git integration reference ....51**
Gerrit directory structure, connectivity and more ....51
TeamForge Git integration with the CollabNet Desktops....54
TeamForge Git integration archives....54
Mappings between TeamForge and Gerrit....54
**TeamForge Git integration release notes....56**
TeamForge Git integration 7.1.4 release notes....56
Fixed issues: TeamForge Git integration 7.1.4....56
TeamForge Git integration 7.1.3 release notes....56
Fixed issues: TeamForge Git integration 7.1.3....56
TeamForge Git integration 7.1.2 release notes....56
Fixed issues: TeamForge Git integration 7.1.2....56
TeamForge Git integration 7.1.0 release notes....57
New features: TeamForge Git integration 7.1.0....57
Fixed issues: TeamForge Git integration 7.1.0....57

# TeamForge Git integration

TeamForge 7.1 supports an integration with the Git distributed version control tool powered by Gerrit.

Although Git is the world's leading distributed version control system, the enterprise has been slow and tentative in its adoption. Concerned with security breaches, compliance violations and lack of governance, many organizations have chosen to take a "wait and see" approach. With TeamForge, Git is ready for the enterprise. TeamForge lets you realize all the benefits of Git while ensuring the security, governance and manageability your business demands. With TeamForge, you can even manage Git and Subversion together, within each individual project.

Gerrit is an open source code review system designed to work with Git. Gerrit supports various access control mechanisms. The TeamForge Git integration uses Gerrit as a vehicle to bring TeamForge project roles and permissions into Git.


### Highlights

- Easy Git repository management from TeamForge using TeamForge's role based access control
- Authentication using SSH keys stored in TeamForge and `http` using Gerrit's `http` passwords
- TeamForge artifact association with Git commits
- Git source code browsing and code search
- Single sign-on between TeamForge and the Gerrit web console
- Full support of Gerrit's APIs to connect with Continuous Integration systems like Jenkins

**Note:** TeamForge 7.1 supports two versions of the Git integration — one based on Gerrit 2.1.10 (version 7.1.x of the integration) and the other based on Gerrit 2.6.x (version 8.0.x of the integration).

### Using this guide

Use this guide for the Git integration based on Gerrit 2.1.10 and supported by TeamForge 7.1:

- *Release notes*
- *Install the Git integration*

For the Git integration based on Gerrit 2.6.x and supported by TeamForge 7.1, see the *TeamForge Git integration - Gerrit 2.6.x* guide.

For releases of the Git integration supported by earlier versions of TeamForge, see these guides:

- *TeamForge 7.0*

- *TeamForge 6.2.x*

**Related Links**

*Git*

*Gerrit*

# Set up the TeamForge Git integration

With TeamForge 7.1, you install the Git integration as part of installing TeamForge.

**Note:** This is different from how you install the Git integration with TeamForge 6.x where you install TeamForge 6.x first, and then follow a separate *process to install the Git integration*.

## Requirements for the TeamForge Git integration

Here's the list of software required for the TeamForge Git integration.

- RedHat Enterprise Linux 6.3 or 6.4
- CentOS 6.3 or 6.4
- SUSE Linux Enterprise Server 11 SP2
- TeamForge 7.1
- JRE 1.6 or later (Oracle JRE only)
- Git 1.7 or later
- PostgreSQL server 9.0.13

  **Note:** You need permissions for the `gerrit` Unix user to connect to the `reviewdb` database from localhost.

- PostgreSQL client
- OpenSSH client
- Apache Web Server with proxy and rewrite modules enabled

In addition, you need to do the following:

- Import the TeamForge host SSL certificate into the Git integration server's JVM trust store
- Import the Git SCM integration server host certificate into the TeamForge server's JVM trust store

**Note:** For a complete list of software supported by TeamForge 7.1, see *these requirements*.

## Install the TeamForge Git integration

You can install the TeamForge Git integration with TeamForge and the Git integration the same server (Local mode) or on different servers (Remote mode).

**RHEL and CentOS**

- To install TeamForge 7.1 and the Git integration on the same server, see *Install TeamForge 7.1 the easy way*.
- To install TeamForge 7.1 on one server and the Git integration on a separate server, see *Install TeamForge 7.1 with the Git integration on a separate server*.

In addition, these installation scenarios are supported:

- *Install TeamForge 7.1 (including the Git integration) on one server and Black Duck Code Sight on a separate server*.
- *Install TeamForge 7.1 on three servers (for intensive database utilization)*.

**SUSE**

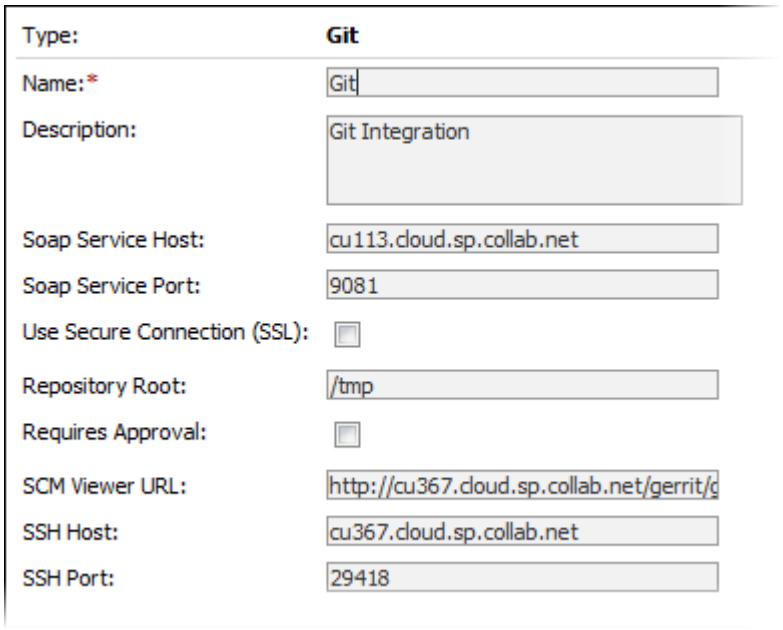
- To install TeamForge 7.1 and the Git integration on the same server, see *Install TeamForge 7.1 the easy way*.

- To install TeamForge 7.1 on one server and the Git integration on a separate server, see *Install TeamForge 7.1 with the Git integration on a separate server*.

In addition, these installation scenarios are supported:

- *Install TeamForge 7.1 (including the Git integration) on one server and Black Duck Code Sight on a separate server*.
- *Install TeamForge 7.1 on three servers (for intensive database utilization)*.

The `post-install.py` script, which you run as part of setting up the Git integration, automatically creates a Git SCM adaptor in TeamForge and sets it up. Here's an example of the **Admin** > **Integrations** > **SCM integrations** page in TeamForge:




**Important:** You can adjust any configuration setting as required, except for **Repository Root**. This parameter is set to "/tmp" — it is important that you NOT change it to some other directory path. It has to be set to this value for backward compatibility reasons and will not affect the actual repository root location in the file system.

Once the post-install script completes successfully, start the `gerrit` service and check the logs, `gerrit-synch.system_log` and `gerrit.system_log` under `/opt/collabnet/gerrit/logs/`, for error messages. In case you see error messages in the logs, you can *reconfigure Gerrit*.

- Code Search functionality is available through integration with Black Duck Code Sight. To enable Black Duck Code Sight for Git, see *Set up Code Search for the TeamForge Git integration* .
- To access the Gerrit web interface directly from TeamForge, you need to *set it up as a linked application*. When that's done, you can log into the Gerrit console and *change access rights (or permissions)*, add internal Gerrit groups and users, and more.

**Related Links**

*TeamForge Git integration: Install FAQ*

## Upgrade the Git integration along with TeamForge

You can upgrade to TeamForge 7.1 with the Git integration on the same server, or with TeamForge and the Git integration on separate servers.

**Special scenario**

Follow the upgrade instructions *here* if either of these conditions is valid for your setup:

- Your Git integration is hosted on a separate, regular Unix server and the corresponding TeamForge host is already updated to version 7.1
- You are using TeamForge 6.2.x or initially configured your Git integration while using TeamForge 6.2.x

**RHEL and CentOS**

- To upgrade to TeamForge 7.1 with the Git integration on the same server, see *Upgrade to TeamForge 7.1 on the same server*.
- To upgrade to TeamForge 7.1 with TeamForge on one server and the Git integration on a separate server, see *Upgrade to TeamForge 7.1 with the Git integration on a separate server*.

**SUSE**

- To upgrade to TeamForge 7.1 with the Git integration on the same server, see *Upgrade to TeamForge 7.1 on the same server*.
- To upgrade to TeamForge 7.1 with TeamForge on one server and the Git integration on a separate server, see *Upgrade to TeamForge 7.1 with the Git integration on a separate server*.

**Related Links**

*TeamForge Git integration: Upgrade and Uninstall FAQ*

## Upgrade the Git integration independent of TeamForge

CollabNet releases versions of the Git integration independent of TeamForge releases. These "point releases" of the integration contain new features, improvements and bug fixes which affect only the Git integration-related components without making changes to the TeamForge server.

Getting point releases is possible only after you have installed (or upgraded to) TeamForge 7.0 (or later) along with the Git integration. To upgrade to a point release independent of TeamForge, follow these directions:

**RHEL and CentOS**

- If you have TeamForge 7.1 and have not initially configured the Git integration with TeamForge 6.2.x, do the following

  ```
  yum upgrade teamforge-git && yum upgrade ctf-git-integration
  ```
  Run `/opt/collabnet/gerrit/scripts/post-install.py`
  Restart `gerrit`: `/etc/init.d/collabnet restart gerrit`

- If you initially configured the Git integration with TeamForge 6.2.x, do the following

  ```
  yum upgrade ctf-git-integration
  ```
  Run `/usr/sbin/ctf-git-integration-setup.sh –-upgrade`.
  Restart `gerrit`: `service gerrit restart`.

**SUSE**

- Whether you have TeamForge 7.1 and the Git integration on the same server or TeamForge 7.1 on one server and the Git integration on an SCM integration server, do the following:

  ```
  zypper update teamforge-git && zypper update ctf-git-integration
  ```
  Run `/opt/collabnet/gerrit/scripts/post-install`.

## Reconfigure the TeamForge Git integration

After you've set up the TeamForge Git integration, it is possible to modify the settings.

- Whether you have TeamForge 7.1 and the Git integration on the same server, or TeamForge 7.1 on one server and the Git integration on a separate SCM integration server, consider the following:
  - Did you change the TeamForge Git integration user's name or password, or the password for the Posgres database "gerrit" role?
  - Did you change properties related to TeamForge or Gerrit in TeamForge's `site-options.conf` file and recreate the runtime?
  - Did you override any Gerrit related parameter directly in `/opt/collabnet/gerrit/etc/gerrit.config`?

  If any of the above is true, do the following to reconfigure the Git integration (on RHEL, CentOS or SUSE):

  Run `/opt/collabnet/gerrit/scripts/post-install.py`.
  Restart `gerrit`: `/etc/init.d/collabnet restart gerrit`

- If you are using TeamForge 6.2.x or initially configured your Git integration while using TeamForge 6.2.x, and have changed a configuration parameter, do the following:

  Run `/usr/sbin/ctf-git-integration-setup.sh`.
  Restart `gerrit`: `service gerrit restart`.

If you want to change the hostname of the TeamForge server or the hostname of the server running the Git integration, take look at the *Post-install FAQ*.

## Uninstall the TeamForge Git integration

To uninstall the integration, use the YUM utility.

Run this command as the `root` or `sudo` user:

```
yum remove ctf-git-integration
```

Uninstalling does not remove your Git repositories from the hard disk. The default location for the repositories is `/gitroot`.

## TeamForge Git integration: VMware image

The VMware image provides the TeamForge Git integration pre-configured, with a default SSH key pair and password already set for the administrator user. If you installed the TeamForge Git integration using the VMware image, we strongly advise you to change both credentials.

### Change the administrator passwords (VMware image)

To change the pre-configured administrator password, specify a new one in the TeamForge **User Details** page and run the post-install script.

The pre-configured version makes use of these accounts and passwords:

- The PostgreSQL `gerrit` user and its password — this password is not exposed

- The TeamForge site administrator account used to connect Gerrit and TeamForge — admin/admin by default

**Note:** If you *set up the integration yourself*, you can specify the passwords for the `gerrit` user as well as the TeamForge user during the install.

If you specify a new Gerrit DB password by running the post-install setup script, you would need to change it in the Gerrit database as well.

- Change the PostgreSQL Gerrit password.

  Log in as `root` in a shell and enter these commands:

```
su postgres
psql
ALTER USER gerrit with password 'your-password';
\q
exit
```

- Run the post-install setup script and change the password to match the one you specified earlier.

  **Note:** Do not change the other default values.

```
service gerrit stop
/usr/sbin/ctf-git-integration-setup.sh
Change gerrit's password [y/N]? y
service gerrit start
```

Follow these steps to change the TeamForge user password:

- Change the password for the TeamForge site administrator.
  a) Click **Admin** in the site navigation bar.
  b) On the site administration navigation bar, click **Users**.
  c) In the list of users, click the name of the administrator.
  d) In the **User Details** page, click **Change Password** and specify the new password.
- Run the post-install setup script and change the password to match the one you specified earlier.

  **Note:** Do not change the other default values.

```
service gerrit stop
/usr/sbin/ctf-git-integration-setup.sh
Change admin's password [y/N]? y
service gerrit start
```

**Related Links**

*Discussion thread on changing the default administrator passwords*

## Change the administrator's SSH key pair (VMware image)

To change the administrator's pre-configured SSH key pair, re-generate the key pair and copy the public key to the TeamForge **Settings** page.

1. Make sure that Gerrit is running on the VMware host.

```
service gerrit check
```

2. Back up the `/opt/collabnet/gerrit/.ssh/` directory.
3. Switch to the `gerrit` system user and run the following commands to regenerate the SSH key pair.

```
su gerrit
ssh-keygen -t rsa
```

**Note:** When you are asked to overwrite the existing SSH key pair in `/opt/collabnet/gerrit/.ssh`, answer "yes"; DO NOT specify a passphrase.

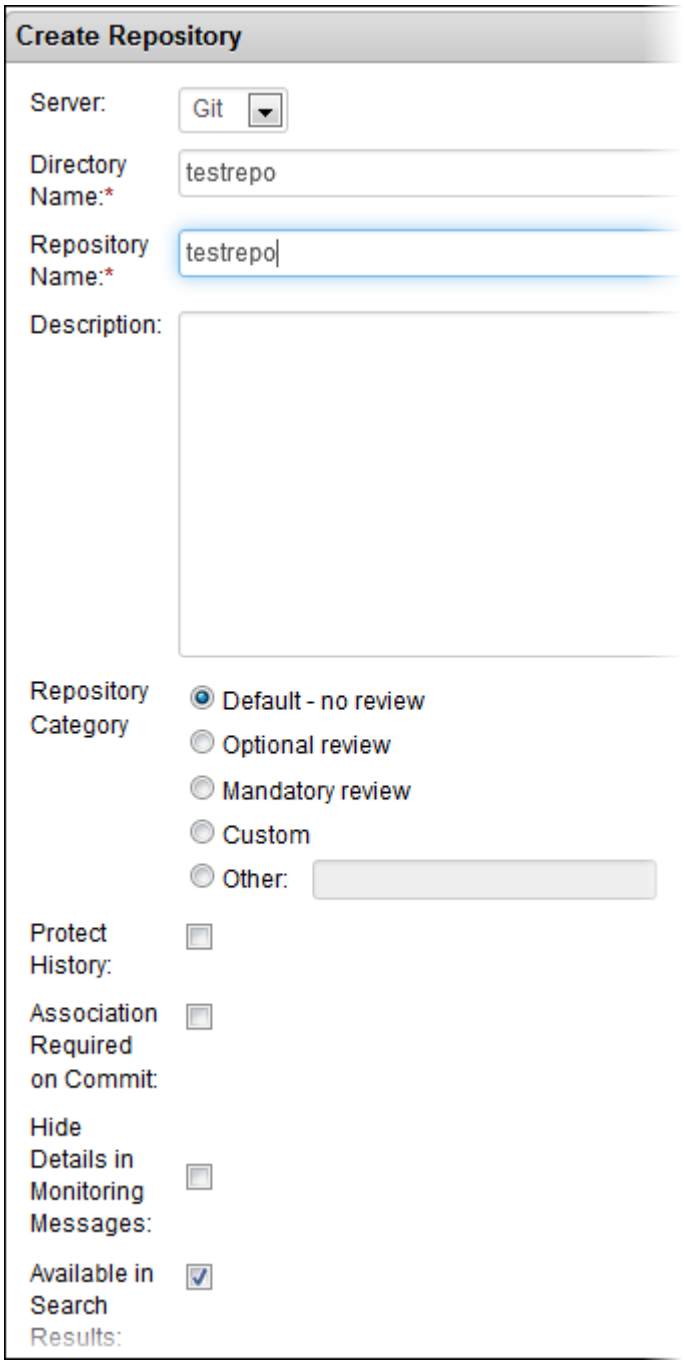
4. Copy the regenerated public key from `/opt/collabnet/gerrit/.ssh/id_rsa.pub` to your clipboard.
5. Using a web browser, log into your TeamForge site as an administrator and do the following:
   a) On your **My Workspace** screen, click **My Settings** in your personal navigation bar.
   b) On your **User Details** page, click **Authorization Keys**.
   c) On the **Authorization Keys** page, remove the existing key(s).
   d) Paste the regenerated public key from your clipboard.
   e) Click **Update**.

# Create a Git repository in TeamForge

After the TeamForge Git integration is installed, project administrators can add Git repositories to TeamForge projects.

1. Click **Source Code** in the project navigation bar in TeamForge.
2. On the page listing the project repositories, click **Create Repository.**

   The **Create Repository** page appears.




3. Choose **Git** for the server on which you want to create the repository.

**Note:** The menu contains all of the SCM servers that the TeamForge administrators have added to the TeamForge environment.

4. Enter the directory name for the repository.
5. Enter a name and description for the repository.
6. Select an option for **Repository Category**.

   This option specifies the code review policy for the repository. For more information, see *Control the code review policy for Git repositories*.
7. Select **Protect History** to turn on history protection for the repository.

   For more information, see *TeamForge Git integration: History protection*.
8. If you want to require that each code commit be associated with an artifact (or a task or some other work item), select **Association Required on Commit**.
9. For security reasons, you may want to restrict email notifications to the essential information. If so, select **Hide Details in Monitoring Messages**.
10. Specify whether you want the repository's source code included in search results.

    By default, the **Available in Search Results** setting is enabled. You may want to de-select this option, for example, if there is sensitive information you want to exclude from search results.
11. Click **Save**.

The repository is automatically synched to Gerrit. The repository name you entered must be unique on the Gerrit server. If so, a new Gerrit project is created with the repository name.

# Control access to Git source code

As a TeamForge project administrator, you can set up roles and permissions to control which users can view or commit Git source code. When you've created a Git repository, add at least one project role with permissions to access that repository.

1. Click **Project Admin** in the project navigation bar.
2. On the **Project Settings** page, click **Permissions**.
3. Select the role for which you want to add Git source code permissions.
   - If the appropriate role does not exist, you must create it first. See *Create a role*.
   - To modify an existing role, click its name.
4. On the **Edit Role** page, click **Source Code**.
5. Specify this role's access to the Git repository.
   - To block users with this role from seeing the repository, select **No Access**.
   - To allow users with this role to see everything in the repository but not commit to it, select **View Only**.
   - To allow users with this role to commit to any path in the repository, select **View and Commit**.
6. Click **Save**.

Make sure you *give team members access to the Git repository* by assigning them the appropriate roles.

# Access Git with an SSH key

To access a Git repository, you need to create a set of SSH authorization keys and use the public-key method for automatic authentication.

Secure shell (SSH) can use public-key cryptography to confirm your identity, authenticate you to the remote host, and encrypt data transmission.

1. Generate an SSH key pair according to the instructions in the documentation for your SSH client.

   (Each SSH client provides its own mechanism for key pair generation.)
2. Log into your TeamForge site.
3. On your **My Workspace** screen, click **My Settings** in your personal navigation bar.
4. On your **User Details** page, click **Authorization Keys**.
5. On the **Authorization Keys** page, copy the public key from the `.pub` file in your SSH installation directory and paste it into the **Authorized Keys** field.
6. Click **Update**.

Your SSH public key is now saved. When you log into a Git repository on your TeamForge site, SSH automatically checks your private key against this public key and authenticates you.

# Clone a Git repository

To get a local copy of a Git repository and start working with the source code, you clone the repository.

To be able to clone a Git repository, you need the following:

- *SSH authorization key*
- *Requisite Source Code permissions*

1. Run the `git clone` command with the URL of the repository. For example:
   ```
   $ git clone ssh://user1@cu001.cloud.collab.net:29418/git_repository
   ```
2. Set your username and email values.
   ```
   $ cd git_repository
   git config --global user.name “user1”
   git config --global user.email “user1@collab.net”
   ```

# Work with Gerrit

When you add Gerrit as a linked application to TeamForge, you can access the web console from TeamForge and update permissions and add internal groups and users.

## Add Gerrit as a linked application

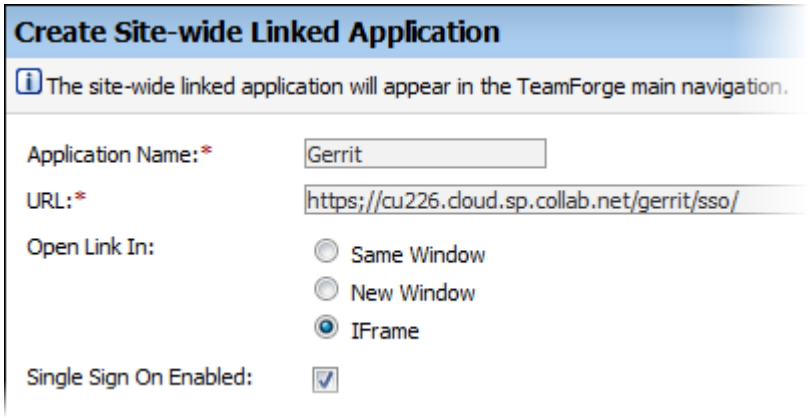
You can set up Gerrit as a linked application to TeamForge at the site or project level.

- Set up the URL `http://<TEAMFORGEHOSTNAME>/gerrit/sso/`.

  **Note:** The last "/" matters. Make sure you have it.

- For instructions on setting up a site-wide linked application, see *Create a site-wide linked application*.

  Here's an example for setting up Gerrit:




  A link for Gerrit is added to your TeamForge navigation bar. Clicking the link displays the Gerrit console in the main TeamForge window.


- To set up Gerrit as a project-wide linked application, see *Link an external application*.
  A button for Gerrit is added to your project navigation bar.

## Control the code review policy for Git repositories

To control how the code in your TeamForge Git repository is reviewed, select the code review policy option for **Repository Category**.

For more information on Gerrit Code Review, see the *Gerrit documentation*.

This feature was introduced in the TeamForge Git integration 7.0.0 release. By default, the following code review policy options are provided:

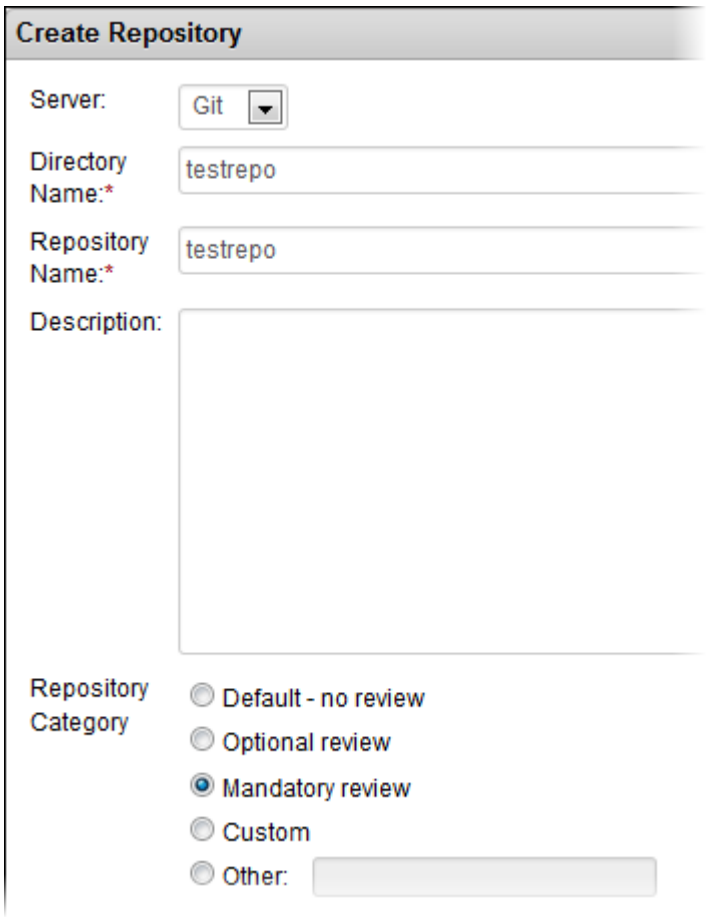
- Default (no code review): All Gerrit review features are turned off, and read/write access is enforced.
- Mandatory review: All code changes must be reviewed, and read/write access is enforced.

- Optional review: The review feature is turned on but can be bypassed if necessary; read/write access is enforced.
- Custom: Access rights must be set manually in the Gerrit web interface; they will not be overridden by TeamForge. This specification is intended for advanced users who are familiar with Gerrit access rights and want to turn off “auto pilot”.

If you have access to the `gerritforge.mappings` file, you can add your own categories.

## Mandatory code reviews for Git repositories

When a mandatory review is specified, every change pushed to the repository must pass through a review process before it can get committed (merged) to the repository.




Only TeamForge users with the **Source Code Admin** permission can bypass reviews.

Here's a list of permissions and what users with these permissions can do:

- No access: Users with no permissions cannot do anything.
- View only: Users with read permissions can read branches and push for reviews, and have -1 and +1 for reviews.
- Commit/View: Users with commit permissions can do everything read permissions would grant and in addition have -2, +2 for reviews. They can verify and submit permissions but have no right to bypass reviews.
- Delete/View: Users with delete permissions can do everything commit permissions would grant.
- Source Code Admin: Users with admin permissions can do everything delete permissions would grant and in addition push to/create any branch (bypassing review). They can rewrite history, forge the identity of the Gerrit server, and have the right to push tags, the right to upload merges, and the right to fine tune access rights in Gerrit for the Gerrit project involved.

**Note:** In TeamForge 6.2, the code review policy for a Git repository is defined in the **Description** field. If you had specified the code review policy in TeamForge 6.2 and have now upgraded TeamForge to version 7.1, you will see the appropriate code review option selected in the TeamForge 7.1 user interface. The **Description** field

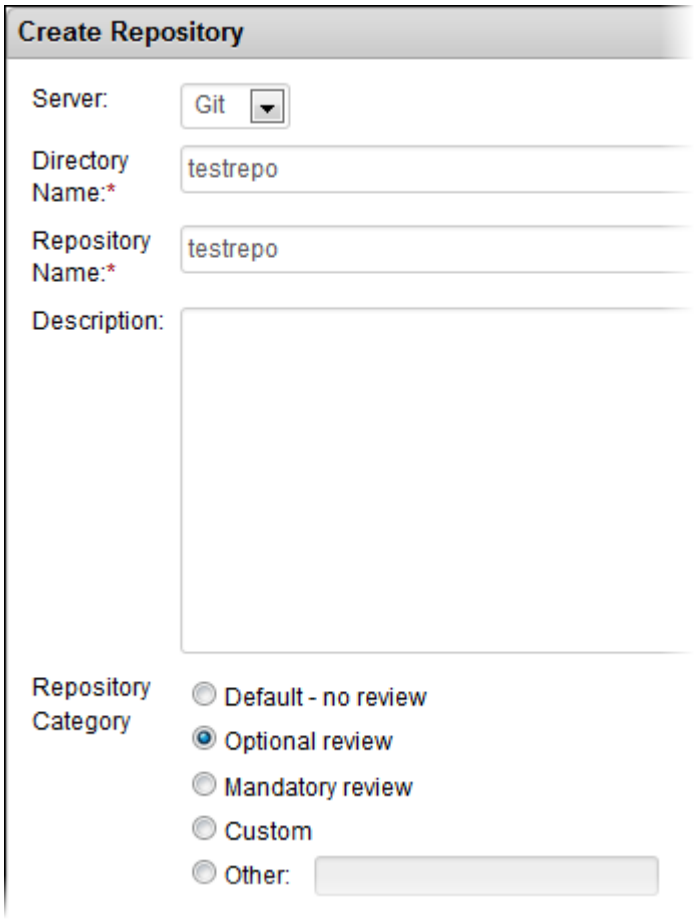
will still display the [RepoCategory:<Category_name>], but you can remove it since it does not have any effect in TeamForge 7.1.

**Related Links**

*TeamForge Git integration: Gerrit Code Review policies FAQ*
*TeamForge Git integration: Gerrit Code Review workflow FAQ*
*Set up a code review process for TeamForge Git repositories*

## Optional code review for Git repositories

When an optional review is specified, every change submitted to the repository can be pushed for code review or directly pushed to the repository bypassing review. This depends on the TeamForge user having the appropriate permissions — source code Delete/View or Commit/View permission for the former, or Source Code Admin permission for the latter.




Here's a list of permissions and what users with these permissions can do:

- No access: Users with no permissions cannot do anything.
- View only: Users with read permissions can read branches and push for reviews, and have -1 and +1 for reviews.
- Commit/View: Users with commit permissions can do everything read permissions would grant and in addition have -2, +2 for reviews. They can verify and submit permissions, push to/create any branch (bypassing review) and push tags.
- Delete/View: Users with delete permissions can do everything commit permissions would grant and in addition, have the right to rewrite history, upload merges and forge identity.
- Source Code Admin: Users with admin permissions can do everything delete permissions would grant and in addition push to/create any branch (bypassing review). They can rewrite history, forge the identity of the Gerrit server, and

have the right to push tags, the right to upload merges, and the right to fine tune access rights in Gerrit for the Gerrit project involved.

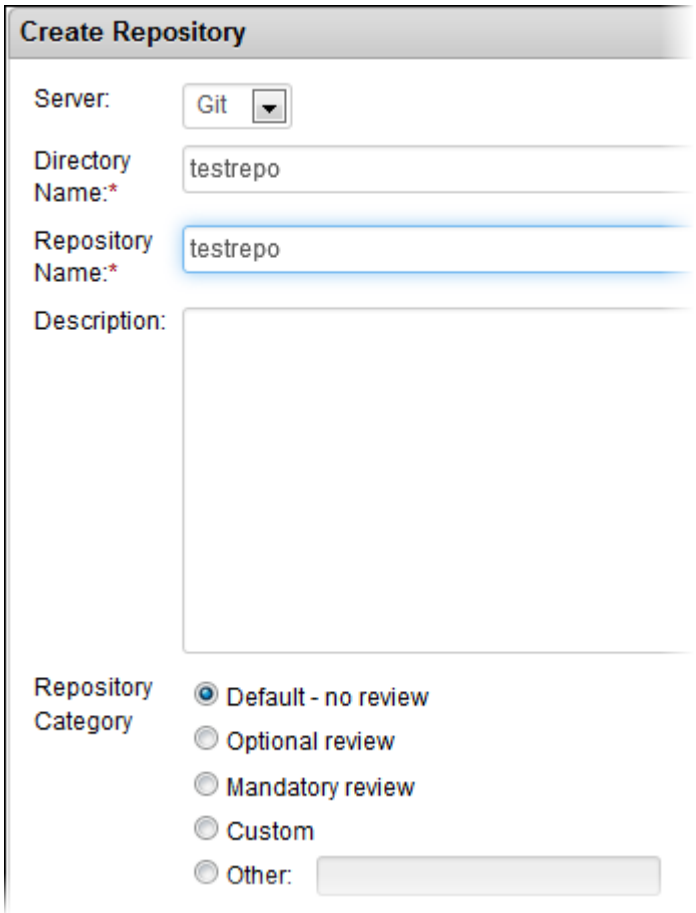
**Note:** In TeamForge 6.2, the code review policy for a Git repository is defined in the **Description** field. If you had specified the code review policy in TeamForge 6.2 and have now upgraded TeamForge to version 7.1, you will see the appropriate code review option selected in the TeamForge 7.1 user interface. The **Description** field will still display the [RepoCategory:<Category_name>], but you can remove it since it does not have any effect in TeamForge 7.1.

**Related Links**

*TeamForge Git integration: Gerrit Code Review policies FAQ*
*TeamForge Git integration: Gerrit Code Review workflow FAQ*
*Set up a code review process for TeamForge Git repositories*

## Default code review for Git repositories

In TeamForge 7.x, the default code review policy — no code review required — is selected unless you choose some other policy.




Here's a list of permissions and what users with these permissions can do:

- No access: Users with no permissions cannot do anything.
- View only: Users with read permissions can only read branches.
- Commit/View: Users with commit permissions can do everything read permissions would grant and in addition, push to/create any branch and push tags.
- Delete/View: Users with delete permissions can do everything commit permissions would grant and in addition, have the right to rewrite history, upload merges and forge identity.

- Source Code Admin: Users with admin permissions can do everything delete permissions would grant. In addition, they can forge the identity of the Gerrit server, and have the right to fine tune access rights in Gerrit for the Gerrit project involved.

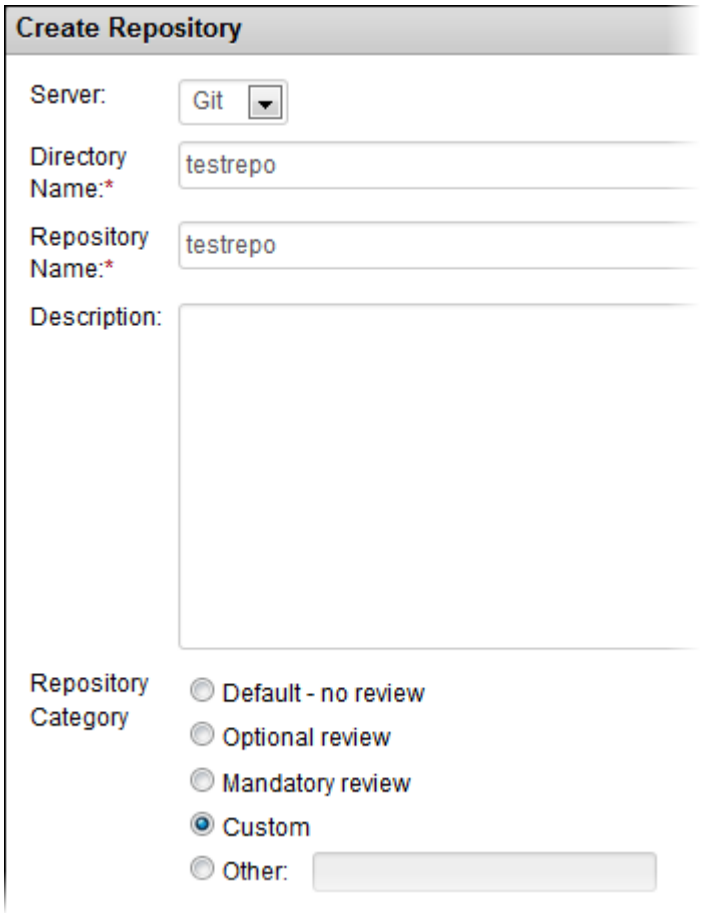
**Note:** In TeamForge 6.2, the code review policy for a Git repository is defined in the **Description** field. If you had specified the code review policy in TeamForge 6.2 and have now upgraded TeamForge to version 7.1, you will see the appropriate code review option selected in the TeamForge 7.1 user interface. The **Description** field will still display the [RepoCategory:<Category_name>], but you can remove it since it does not have any effect in TeamForge 7.1.

**Related Links**

*TeamForge Git integration: Gerrit Code Review policies FAQ*
*TeamForge Git integration: Gerrit Code Review workflow FAQ*
*Set up a code review process for TeamForge Git repositories*

## Custom code review for Git repositories

When a custom code review is specified, users with the TeamForge Source Code Admin permission can directly fine tune permissions (access rights) in Gerrit's web interface. Those changes will not be overridden by TeamForge.




For information on manually defining access rights in the Gerrit web interface, see *Update Git repository access permissions in Gerrit*.

Here's a list of permissions and what users with these permissions can do:

- No access: Users with no permissions cannot do anything.
- View only: Users with read permissions cannot do anything unless added in Gerrit.
- Commit/View: Users with commit permissions cannot do anything unless added in Gerrit.

- Delete/View: Users with delete permissions cannot do anything unless added in Gerrit.
- Source Code Admin: Users with admin permissions have the right to fine tune access rights in Gerrit for the Gerrit project involved.

In case you decide to change the **Repository Category** value to **Custom** from some other value, current access rights will not be affected. Subsequent changes to source code permissions assigned to project roles in the corresponding TeamForge project will have no impact in Gerrit. Basically, defining a custom review is like turning off “auto pilot”.

**Note:** In TeamForge 6.2, the code review policy for a Git repository is defined in the **Description** field. If you had specified the code review policy in TeamForge 6.2 and have now upgraded TeamForge to version 7.1, you will see the appropriate code review option selected in the TeamForge 7.1 user interface. The **Description** field will still display the [RepoCategory:<Category_name>], but you can remove it since it does not have any effect in TeamForge 7.1.

**Related Links**

*TeamForge Git integration: Gerrit Code Review policies FAQ*

*TeamForge Git integration: Gerrit Code Review workflow FAQ*

*Set up a code review process for TeamForge Git repositories*

## Code review policies: mappings

This table shows how source code permissions in TeamForge are mapped to access rights in Gerrit based on the code review policy.

| TF Cluster / Repo Category | None | SCM View Only | SCM Commit / View | SCM Delete / View | SCM Admin |
|---|---|---|---|---|---|
| default | READ,-1,-1 | READ,1,1 | READ,1,1<br>pHD ,1,2<br>pTAG,1,2,refs/tags/* | READ,1,3<br>pHD ,1,3<br>pTAG,1,2,refs/tags/*<br>FORG,1,2 | READ,1,3<br>pHD ,1,3<br>pTAG,1,2,refs/tags/*<br>FORG,1,3<br>OWN ,1,1 |
| mandatory_review | READ,-1,-1 | READ,1,2<br>CRVW,-1,1 | READ,1,2<br>CRVW,-2,+2<br>VRIF,-1,1<br>SUBM,1,1 | READ,1,2<br>CRVW,-2,+2<br>VRIF,-1,1<br>SUBM,1,1 | READ,1,3<br>CRVW,-2,+2<br>VRIF,-1,1<br>SUBM,1,1<br>pHD ,1,3<br>pTAG,1,2,refs/tags/*<br>FORG,1,3<br>OWN ,1,1 |
| optional_review | READ,-1,-1 | READ,1,2<br>CRVW,-1,1 | READ,1,2<br>CRVW,-2,+2<br>VRIF,-1,1<br>SUBM,1,1<br>pHD ,1,2<br>pTAG,1,2,refs/tags/* | READ,1,3<br>CRVW,-2,+2<br>VRIF,-1,1<br>SUBM,1,1<br>pHD ,1,3<br>pTAG,1,2,refs/tags/*<br>FORG,1,2 | READ,1,3<br>CRVW,-2,+2<br>VRIF,-1,1<br>SUBM,1,1<br>pHD ,1,3<br>pTAG,1,2,refs/tags/*<br>FORG,1,3<br>OWN ,1,1 |
| custom | - | - | - | - | OWN ,1,1 |

## Notes on customizing your code review policy

To modify an existing code review policy or create your own policy, you need to update the `/opt/collabnet/gerrit/etc/gerritforge.mappings` file. This requires file system access to the Git integration server.

While you are experimenting with the settings, we recommend that you create a new code review policy rather than modify an existing one. If you delete the `gerritforge.mappings` file from the file system, it will be automatically recreated with default settings after Gerrit is restarted.

`gerritforge.mappings` is a Java property file with the format [<name of Git Repo Category>.]<TeamForge SCM permission cluster>.<number of entry>=<Gerrit Access Right Category Code>,(<Lower Bound>,<Upper Bound>[,<ref spec>].

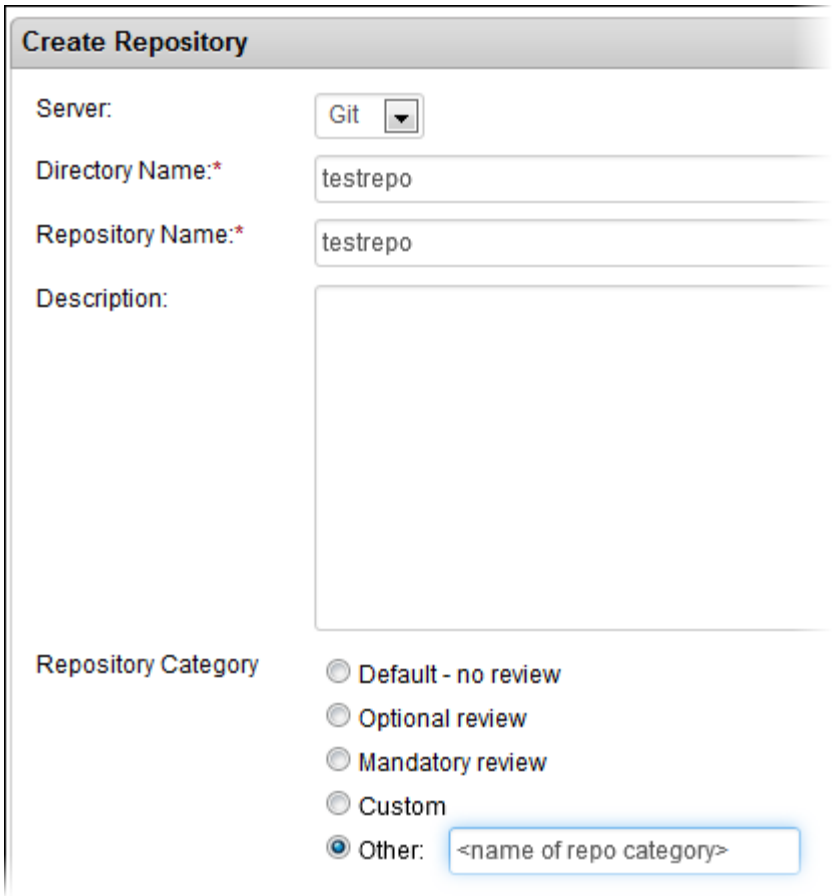
- Git Repo Category is the term internally used to describe a code review policy.
- If no `ref spec` is specified, the default value `refs/*` will be assumed.
- If a TeamForge project role has multiple permission clusters, only the most powerful one (scm_admin > scm_delete > scm_commit > scm_view > none) mentioned in the mapping rules file for the corresponding repository category

will be considered. If none of the permission clusters of a TeamForge project role are mentioned in the mapping rules file,"none" will assumed. If "none" is not mentioned in the mapping rules, no access rights will be assumed.

- If a Gerrit access right category code is mentioned for a repository category, all previously existing access rights of that access right category will be replaced as long as the `[<name of Git Repo Category>.]keep_rights_added_in_gerrit` property is set to "false". If this property is set to "true", existing rights will be kept. The `[<name of Git Repo Category>.]keep_rights_added_in_gerrit` property also determines whether access rights of Gerrit categories not explicitly mentioned for the repository category will be deleted or kept.

Whenever a decision about an access right mapping has to be made for a TeamForge Git repository (which corresponds to a Gerrit project), the repository description is parsed for a string matching the pattern "[RepoCategory:<name of repo category>]". If no occurrence is found, default mapping rules (properties without a repository category) will be used; otherwise, the properties starting with <name of repo category> of the first match will be used. If the repository category is not found, a warning will be issued in the logs and the access rights currently defined will be preserved.

In TeamForge 7.x, you can specify the name of a code review policy that you've defined or customized in the **Other** text field:




**Note:** In TeamForge 6.2, the code review policy for a Git repository is defined in the **Description** field. If you had specified the code review policy in TeamForge 6.2 and have now upgraded TeamForge to version 7.1, you will see the appropriate code review option selected in the TeamForge 7.1 user interface. The **Description** field will still display the [RepoCategory:<Category_name>], but you can remove it since it does not have any effect in TeamForge 7.1.

## Set up a code review process for TeamForge Git repositories

Decide whether you want code review for your project's Git repository to be mandatory or optional and then follow this general process.

This feature is available with version 7.0.0 (and later) of the TeamForge Git integration.

- To enforce code reviews, do the following:
  a) As the project administrator, select the TeamForge Git repository for which you want to enforce code review.
  b) For **Repository Category**, select the **Mandatory review** option and save it.
  c) Set up the following roles:
     - Create a "developer" role. This user can access source code in read only mode and send code for review. This role has **View only** permissions for the Git repository.
     - Create a "reviewer" role. This user can access source code, review code, approve or reject changes, and decide whether to submit a change to the central Git repository in TeamForge. This role has project-level **Delete/View** and **Commit/View** source code permissions.

  With these changes in place, a user with the "developer" role can check out code, change code and send it for review. A user with the "reviewer" role can review a change and ultimately decide whether to submit the change to the central Git repository.
- To keep code reviews optional, do the following:
  a) As the project administrator, select the TeamForge Git repository for which you want to set up the code review policy.
  b) For **Repository Category**, select the **Optional review** option and save it.
  c) Set up the following roles:
     - Create a "developer" role. This user can access source code in read only mode and send code for review. This role has **View/Commit** permissions for the Git repository.
     - Create a "reviewer" role. This user can access source code, review code, approve or reject changes, and decide whether to submit a change to the central Git repository in TeamForge. This role has project-level **Delete/View** and **Commit/View** source code permissions.

  With these changes in place, a user with the "developer" role can check out code, change code and (optionally) send it for review. A user with the "reviewer" role can review a change and ultimately decide whether to submit the change to the central Git repository. Alternatively, since the review process is optional, the "developer" user can push changes to the central Git repository.

**Related Links**

*TeamForge Git integration: Gerrit Code Review policies FAQ*
*TeamForge Git integration: Gerrit Code Review workflow FAQ*
*Mandatory code reviews for Git repositories*
*Optional code review for Git repositories*
*Default code review for Git repositories*
*Custom code review for Git repositories*

## Prepare the Git client for code review

Gerrit Code Review provides a standard `commit-msg` hook which you can install in your local Git repository to automatically create and insert a unique Change-Id line to a Git commit message.

When a change is reworked, the Change-Id associates the new version back to the original review.

To get the hook into your local Git repository, run this command:

```
scp -p -P 29418 <user>@<GerritHostServer>:hooks/commit-msg .git/hooks
```

**Related Links**

*`commit-msg` hook*
*Change-Ids*
*Gerrit Code Review - Change-Ids: Creation*

## TeamForge Git integration: Gerrit Code Review policies FAQ

Questions on using Gerrit Code Review for Git repositories in TeamForge.

**What is Gerrit Code Review? What does it have to do with the TeamForge Git integration?**

Gerrit Code Review is an open source project for code review and verification for the Git version control system. It also has capabilities for administering Git repositories and controlling access to source code. The TeamForge Git integration provides TeamForge's role-based access control (RBAC) feature to Gerrit and thus to Git repositories hosted by TeamForge.

**Is it possible to use Gerrit Code Review when my TeamForge project has Git repositories?**

Yes, you can turn on the Gerrit Code Review feature for Git repositories in your TeamForge project. By default, this feature is switched off when you create a Git repository. You can turn it on or off for individual repositories from your TeamForge project configuration.

**In what ways can I use the code review process in my TeamForge project's Git repository?**

There are several possibilities. You can —

- Choose to not use the code review feature
- Enforce code review for every commit made to a TeamForge Git repository and control the review process (such as who can review, who can vote for a code review and so on) from TeamForge
- Use code review but have the possibility to bypass it, and also control the review process from TeamForge
- Control the review process entirely from Gerrit (not TeamForge)

For more information, see *Control the code review policy for Git repositories*.

**Can I enforce code review for a TeamForge Git repository but handle the review control process in Gerrit?**

Yes. You can enforce code review and have the have review process control defined in Gerrit without using TeamForge's role-based acces contriol for source code. In the TeamForge Git repository's description field, include the string [RepoCategory:custom]. When you do that, TeamForge source code permissions for that project will no longer apply. You can define your own internal group in Gerrit and give it appropriate permissions for the Gerrit project corresponding to the Git repository in TeamForge. For more information, see *Manage Gerrit access rights outside TeamForge*.

**Can I enable Gerrit code review for individual Git repositories in TeamForge?**

Yes. By default, Gerrit code review is disabled when you create a new Git repository in your TeamForge project. To enable code review, you modify the description field of the repository suitably.

**I just changed the repository description to impose a code review policy for my Git repository. Why is the Review button not enabled when I click on a review request in the Gerrit web interface?**

If you changed the repository description as a user with the Source Code Admin permission, the change will not get reflected in Gerrit immediately. In case you want this change in effect almost immediately, the following needs to happen: soon after you change the repository description, the TeamForge project administrator must remove any user with the Source Code Admin permission temporarily, and then add that user back. This action will trigger an immediate sync which will enable the code review feature. You will then see the Review button enabled for any review in the Gerrit web interface. If you don't do this, the code review policy will apply after a periodical sync.

**Related Links**

*Set up a code review process for TeamForge Git repositories*
*Mandatory code reviews for Git repositories*

*Optional code review for Git repositories*
*Default code review for Git repositories*
*Custom code review for Git repositories*

### TeamForge Git integration: Gerrit Code Review workflow FAQ

Questions on using Gerrit Code Review for Git repositories in TeamForge.

| | |
|---|---|
| **What is the typical workflow when you use Gerrit Code Review and TeamForge?** | Please see the *Getting Started with Gerrit* guide for details. |
| **How does the TeamForge Git Integration manage access rights in Gerrit for the code review process?** | The TeamForge Git integration maps TeamForge's role-based access control (RBAC) features to Gerrit access rights. |
| **How can I integrate the Jenkins Continuous Integration server with the TeamForge Git integration?** | Take a look at *this blog* for details. |
| **If there is no Jenkins setup, who can set "verified" for a review request?** | If Jenkins is not integrated, a TeamForge user having a role with the Source Code Admin or SourceCcode Delete/View plus Commit/View permissions can manually set "verified". |
| **What if Jenkins sets "verified fail (-1)": can some other user override "verified" for a review request?** | No, some other user (who is able to set "verified" for a review request) cannot be override this; only Jenkins or the user who set "verified" can do this. However, if a review request is amended, some other user can set "verified". |
| **Is it possible to discard a review request?** | Yes; but only the user who submitted the review request can abandon that request, nobody else can do that. |

**Related Links**

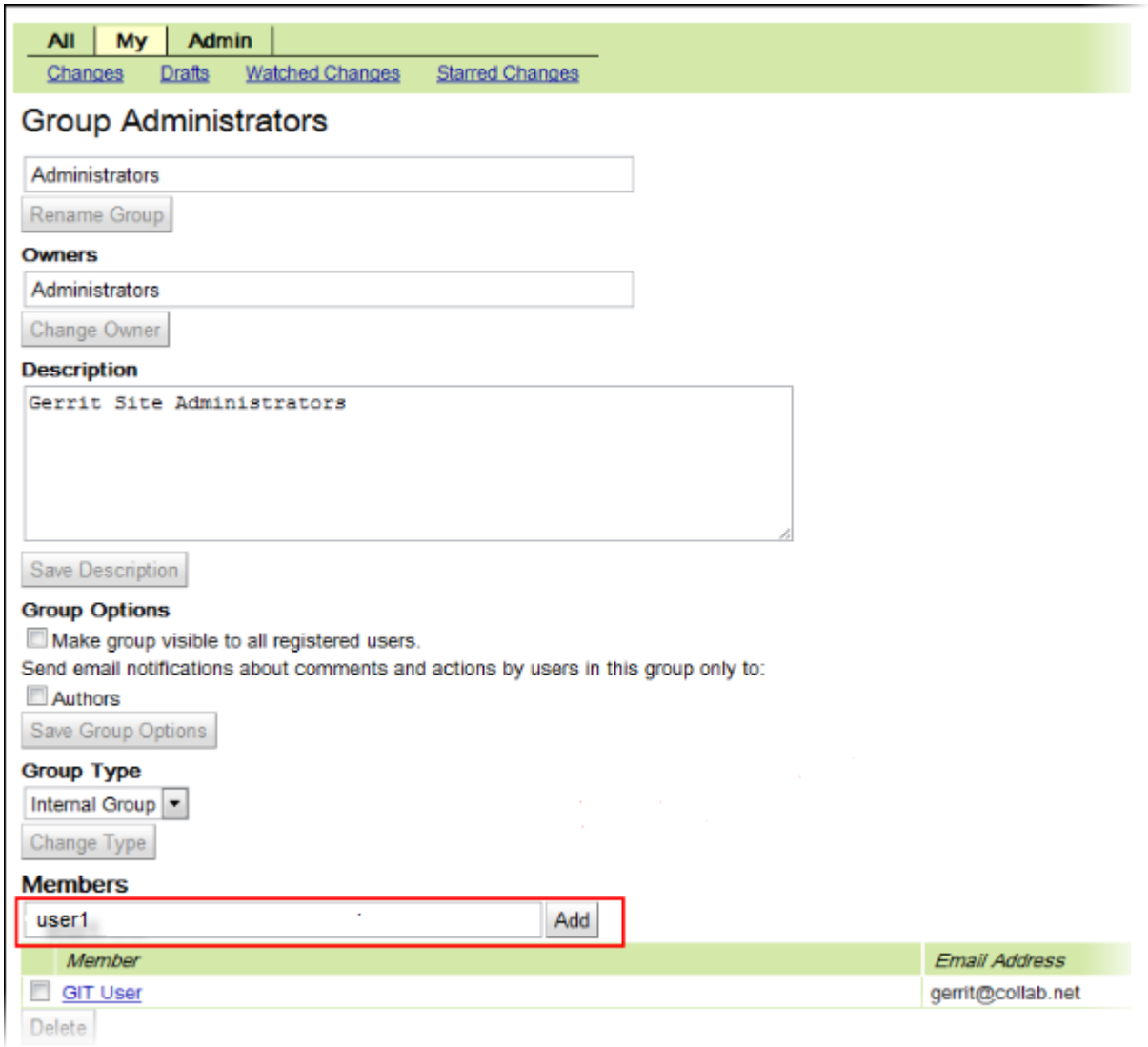
*Set up a code review process for TeamForge Git repositories*
*Mandatory code reviews for Git repositories*
*Optional code review for Git repositories*
*Default code review for Git repositories*
*Custom code review for Git repositories*

## Add a TeamForge user to Gerrit Administrators

To grant additional TeamForge users Gerrit super user permissions, add the users to the Administrators group.

1. In your browser, bring up `http://<GITSCMSERVERHOSTNAME>/gerrit/`.
2. Log into Gerrit using the TeamForge site administrator username and password you provided while running the Git installer's configuration script.
3. In the My tab, click **Settings**.
4. In the left panel, click **Groups** and select **Administrators**.


5. Click **Administrators** and add new members to the group.



The TeamForge users you add to the Administrators group will have Gerrit super user permissions. These users do not have to be site administrators in TeamForge.

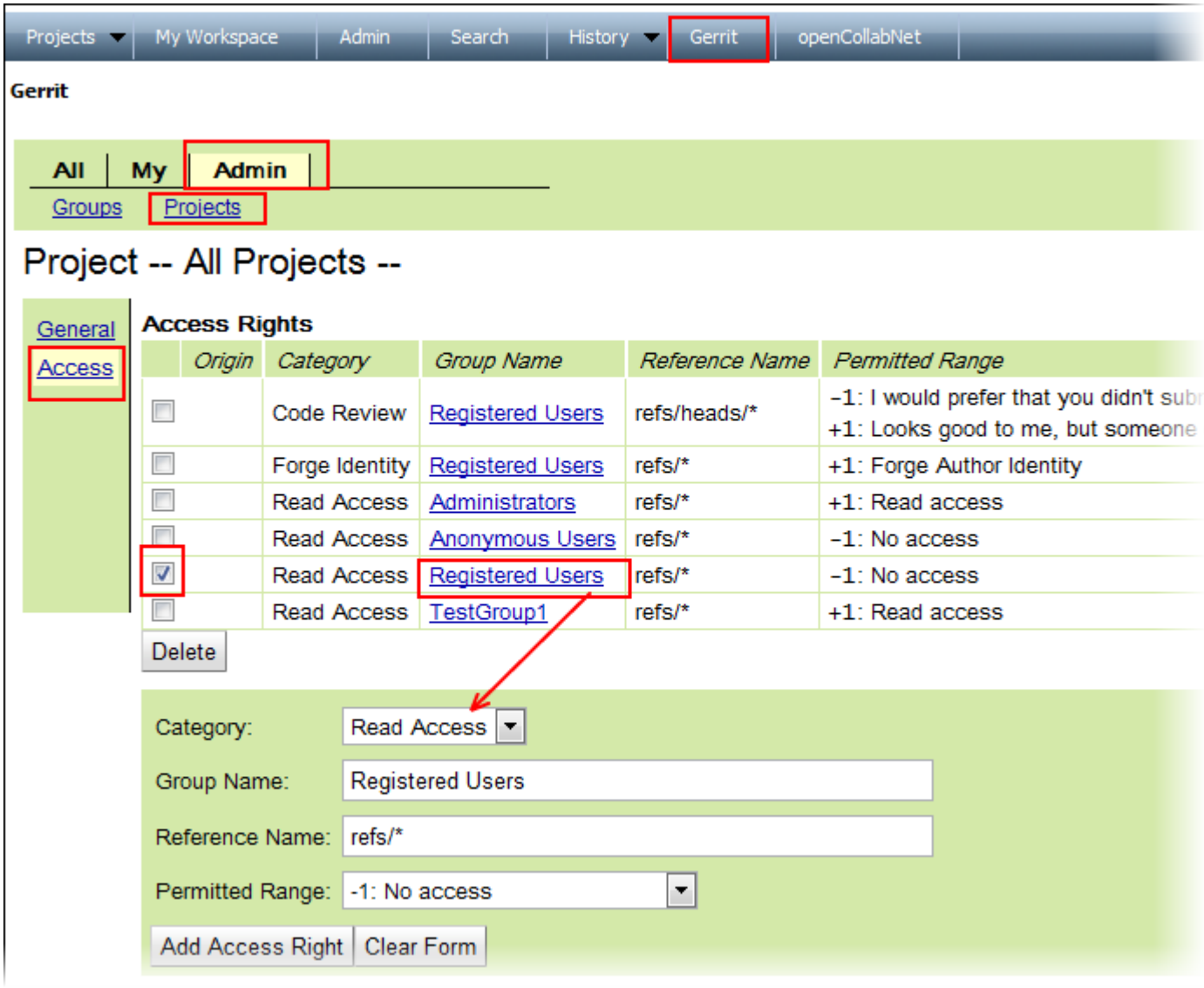
**Tip:** If you want to assign additional permissions to a group of users, make use of Gerrit's internal group feature. See *Set up Code Search for the TeamForge Git integration* for an example of how you can use this feature to grant a group of users read permissions to all Git repositories.

## Update Git repository access rights in Gerrit

By default, Gerrit projects (TeamForge Git repositories) are only visible to TeamForge users assigned a project role with SCM permissions. To grant additional rights (for example, read access) to all registered users, or to TeamForge global groups, or to a custom subset of users (a Gerrit internal group), log directly into the Gerrit console and make those changes.

1. In your browser, bring up `http://<GITSCMSERVERHOSTNAME>/gerrit/`.
2. Log into Gerrit using the TeamForge site administrator username and password you provided while running the Git installer's configuration script.
3. Select the Admin tab and click **Projects**.
4. In the **All Projects** page, click **Access**.
   You will see a list of all default access rights.
5. To change an access right, select a group and specify its category.

   Here's an example where the group "Registered Users" is assigned the category "Read Access".




6. Click **Add Access Right**.

## Manage Gerrit access rights outside TeamForge

When existing TeamForge user groups or project roles doesn't satisfy your requirements for organizational or other reasons, you can add and manage access rights directly in Gerrit, bypassing the source code permissions in TeamForge. You do this by creating an internal group in Gerrit and assigning this group appropriate access rights for the code and the code review process.

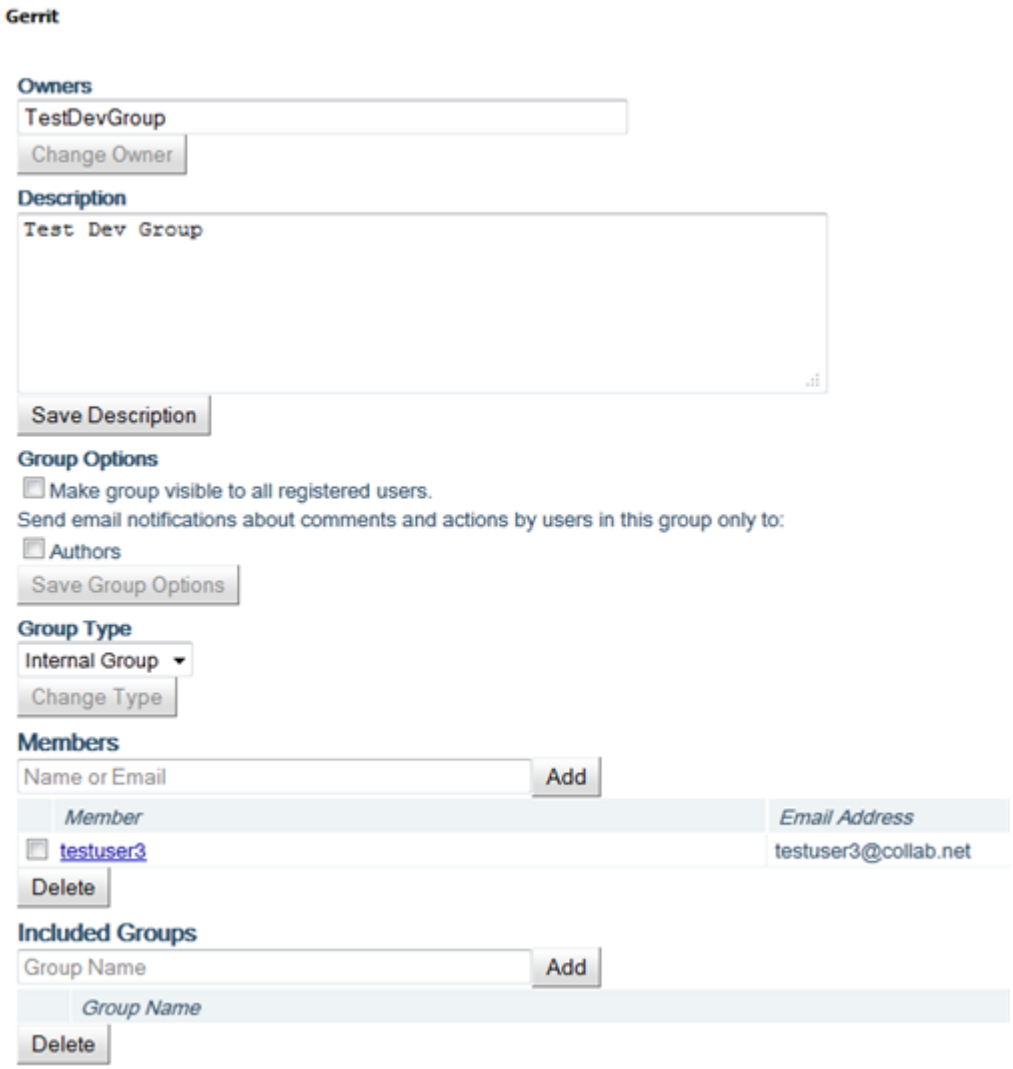
This feature is available with the TeamForge Git integration version 7.0.0 (and later).

**Note:** In most situations, using the custom category in TeamForge or defining your own `RepoCategory` is better than defining internal Gerrit groups. See *this blog post* for more information.

1. In the TeamForge project with the Git repository, make sure you have a role with the **Source Code Admin** permission and there is a user assigned this role.
2. Log into the Gerrit web interface as the above user by doing one of the following:
   - Click the Gerrit tab in TeamForge.

- Type in the URL `http://<TEAMFORGE HOSTNAME>/`.

3. In Gerrit, select the **Admin** tab and click **Groups**.
4. Create a new group and provide it a description.



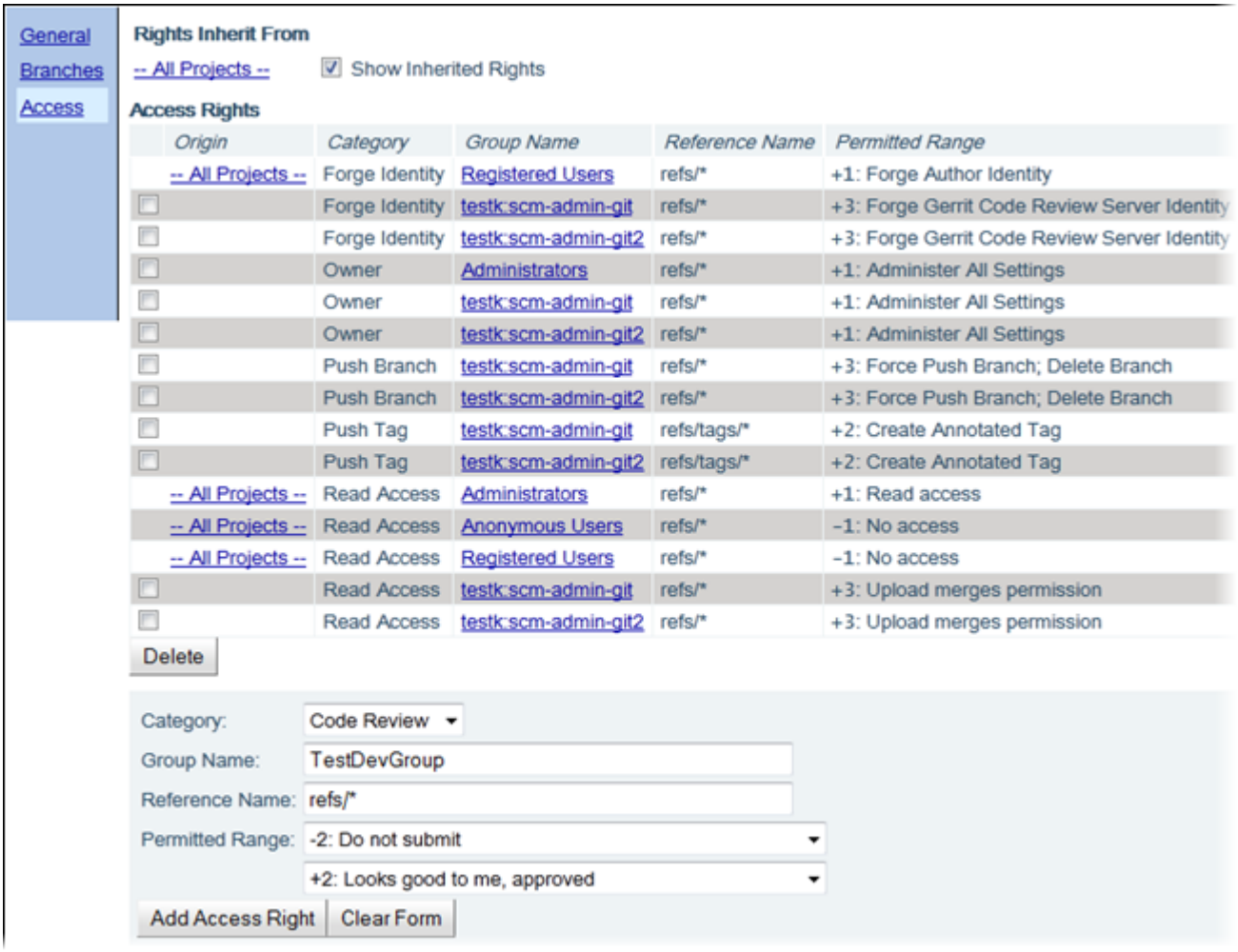

You can now do the following:

a) In the **Group Type** dropdown, select Internal Group.
b) Add TeamForge users in the **Members** field.
c) Add TeamForge user groups in **Included Groups**.

   Type in ":" followed by the user group name. All members of this user group will automatically receive all rights assigned for this Gerrit internal group.

5. To provide this group access rights in the Gerrit project (corresponding to the Git repository in TeamForge), follow these steps:

a) In the **Admin** tab, click **Projects** and then click the project for which you want to specify access rights.
b) Click **Access**, and specify the category, group, reference (for example, `refs/*`, `refs/<BRANCHNAME>`), and range.




c) Click **Add Access Right**.

You can provide the group multiple access rights. All members and included groups will now have the appropriate access rights for the Gerrit project (Git repository in TeamForge). For more information, see the Gerrit help on *access rights and permissions*.

In case, you want to tightly control who can change information for a newly created internal group, then you, as the Gerrit site administrator, must set the "Administrator" group (a default system group having a Gerrit site administrator as initial member) in the **Owners** field. For more information, see the Gerrit help on *System Groups*.

## Set up Code Search for the TeamForge Git integration

To use TeamForge Code Search functionality for Git, manually grant the TeamForge Code Search user permissions to access all Git repositories.

In TeamForge 6.2 (and later versions), Code Search functionality is available through integration with Black Duck Code Sight.

To be able to access the Gerrit console directly from TeamForge, you'll need to *set it up as a linked application*.

1. Make sure that a Unix system user with the name `scmviewer` exists. If not, create this user by running the following command:

```
useradd scmviewer
```

2. You need the root user's public key for SSH authentication on the Code Search server. Do the following:
   a) On the Code Search box, check whether the key is present at `/root/.ssh/id_rsa.pub`. If not, generate it by running the `ssh-keygen` command.
   b) Copy it to a temporary location (`/tmp`) on the TeamForge application server.
   c) Run the *`set_auth_key.py`* script for the `scmviewer` user on the TeamForge application server.

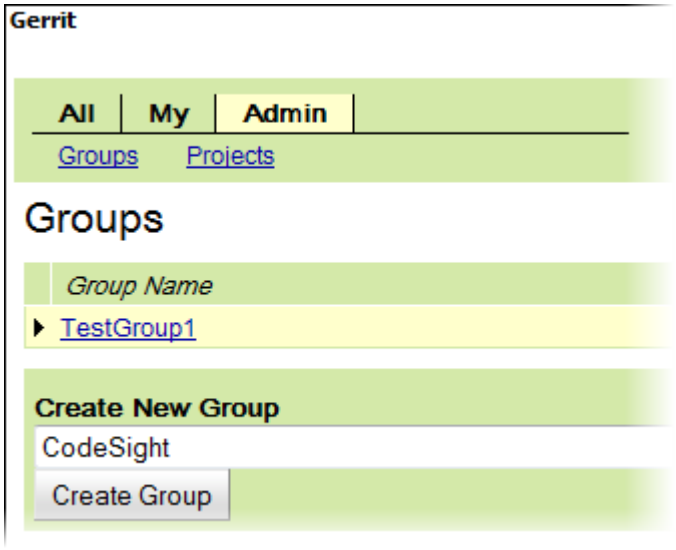
   **Note:** This script requires TeamForge site administrator credentials.

   ```
   cd /opt/collabnet/teamforge/runtime/scripts/codesearch/
   ./set_auth_key.py --authkey-file=/tmp/id_rsa.pub
   ```

3. Sync the `scmviewer` user to Gerrit by running the following command in a shell on the host where you installed the Git integration.

   ```
   curl http://localhost:9081/api/gerrit/users/scmviewer/sshkeys
   ```

4. Log into the Gerrit console as a Gerrit super user and create an internal Gerrit group, for example, "CodeSight Group".




5. Add the `scmviewer` user to the group.
6. Grant read access to the group.
   a) In the Gerrit project page that displays access rights, select **Read Access** for Category.
   b) For **Group Name**, enter the name of the internal group ("CodeSight" in the example) you created.
   c) Enter "refs/*" for **Reference Name**.
   d) Enter "+1:Read Access" for **Permitted Rage**.
   e) Click **Add Access Right**.
7. Log out from Gerrit.
8. Restart the Code Search server.

   ```
   /etc/init.d/collabnet restart tomcatcs
   ```

# Gerrit configuration (for advanced users)

Here are some recommendations for `gerrit.config` settings.

## Memory settings in Gerrit configuration

Memory settings in `gerrit.config`.

### heapLimit

Memory is crucial to Gerrit performance. The `container.heapLimit` parameter is used here. From the *Gerrit documentation:*

```
container.heapLimit
    Maximum heap size of the Java process running Gerrit, in bytes. This property
 is translated into the -Xmx flag for the JVM.
    Default is platform and JVM specific.
    Common unit suffixes of k, m, or g are supported.
```

Our default configuration does not set this parameter. So what should the default value be? According to Oracle documentation, starting from J2SE 5.0 it is the smaller of 1/4th of the physical memory or 1GB (before J2SE 5.0, the default maximum heap size was 64MB). We recommend you set this parameter to 8GB for a 16GB machine and 16GB or more for a machine with 24GB or more memory.

### packedGitLimit

This is another important setting. From the *Gerrit documentation*:

```
core.packedGitLimit
    Maximum number of bytes to load and cache in memory from pack files.
    If JGit needs to access more than this many bytes it will unload less
frequently used windows to reclaim memory space within the process.
    As this buffer must be shared with the rest of the JVM heap, it should be a
 fraction of the total memory available.
    Default on JGit is 10 MiB on all platforms.
    Common unit suffixes of k, m, or g are supported.
```

From *this post* on Gerrit performance:

```
We run an 8 GiB JVM heap, and of that heap we permit 2 GiB of memory to used as
 a buffer cache for packed Git data.
By default that buffer cache is 20 MiB.  If you don't raise packedGitLimit that
 JVM heap won't really be utilized and you
will be thrasing the internal buffer cache.
```

So it is good to increase `packedGitLimit` to at least 2GB. This is in keeping with the experience of our customers who use Gerrit heavily. We recommend you set this parameter to 2GB or 4GB.

### packedGitWindowSize

```
core.packedGitWindowSize
   Number of bytes of a pack file to load into memory in a single read operation.
 This is the "page size" of the JGit buffer cache,
    used for all pack access operations. All disk IO occurs as single window
reads. Setting this too large may cause the process
    to load more data than is required; setting this too small may increase the
 frequency of read() system calls.
    Default on JGit is 8 KiB on all platforms.
    Common unit suffixes of k, m, or g are supported.
```

Suggested setting:

```
core.packedGitWindowSize = 16k (default is 8k)
```

#### packedGitOpenFiles

```
core.packedGitOpenFiles
    Maximum number of pack files to have open at once. A pack file must be opened
 in order for any of its data to be available in a cached window.
    If you increase this to a larger setting you may need to also adjust the
ulimit on file descriptors for the host JVM, as Gerrit needs additional file
    descriptors available for network sockets and other repository data
manipulation.
    Default on JGit is 128 file descriptors on all platforms.
```

Suggested setting:

```
core.packedGitOpenFiles = 4096 (default is 128)
```

**Note:** Changing this parameter may require adjusting ulimit on file descriptors for the host JVM.

## sshd settings in Gerrit configuration

`sshd` settings in `gerrit.config`.

#### sshd.threads

This is another setting crucial to Gerrit performance. From *Gerrit documentation*:

```
Number of threads to use when executing SSH command requests.
If additional requests are received while all threads are busy they are queued
and serviced in a first-come-first-serve order.
By default, 1.5x the number of CPUs available to the JVM.
```

Based on our customer experience, you should set this parameter between 24 and 56. We recommend 48 as initial value and then fine-tuning it.

#### sshd.batchThreads

```
Number of threads to allocate for SSH command requests from non-interactive
users.
If equals to 0, then all non-interactive requests are executed in the same queue
 as interactive requests.
Any other value will remove the number of threads from the queue allocated to
interactive users,
and create a separate thread pool of the requested size, which will be used to
run commands from non-interactive users.
If the number of threads requested for non-interactive users is larger than the
 total number of threads allocated in sshd.threads,
then the value of sshd.threads is increased to accomodate the requested value.
By default, 0.
```

Value used by one of our customers:

```
sshd.batchThreads = 2
```

#### sshd.streamThreads

```
Number of threads to use when formatting events to asynchronous streaming clients.

Event formatting is multiplexed onto this thread pool by a simple FIFO scheduling
 system.
By default, 1 plus the number of CPUs available to the JVM.
```

Value used by one of our customers:

```
sshd.streamThreads = 50
```

**sshd.maxAuthTries**

```
Maximum number of authentication attempts before the server disconnects the
client.
Each public key that a client has loaded into its local agent counts as one auth
 request.
Users can work around the server's limit by loading less keys into their agent,

or selecting a specific key in their ~/.ssh/config file with the IdentityFile
option.
By default, 6.
```

Value used by one of our customers:

```
sshd.maxAuthTries = 12
```

## Database settings in Gerrit configuration

Database settings in `gerrit.config`.

**database.poolLimit**

You might want to tune the *`database.poolLimit`* parameter:

```
Maximum number of open database connections. If the server needs more than this
 number,
request processing threads will wait up to poolMaxWait seconds for a connection
 to be released
before they abort with an exception. This limit must be several units higher
than the total number
of httpd and sshd threads as some request processing code paths may need multiple
 connections.
Default is 8.
```

In the `gerrit.config` file, it is set by default to 50 which is a reasonable value to start with. As a reference, here are some other values used by our customers: 52, 64 and 250.

**database.poolMaxIdle**

```
Maximum number of connections to keep idle in the pool.
    If there are more idle connections, connections will be closed
    instead of being returned back to the pool. Default is 4.
```

Values used by our customers: 12, 16

**database.poolMaxWait**

```
Maximum amount of time a request processing thread will wait to acquire a database

connection from the pool. If no connection is released within this time period,
 the processing
thread will abort its current operations and return an error to the client.
Values should use
common unit suffixes to express their setting:

ms, milliseconds
s, sec, second, seconds
m, min, minute, minutes
h, hr, hour, hours
If a unit suffix is not specified, milliseconds is assumed.
Default is 30 seconds.
```

Values used by our customers: 60 seconds

### `log4j` settings in Gerrit configuration

`log4j` settings in `gerrit.config`.

For heavy users, we recommended switching from verbose tracing to normal logging in the `log4j.properties` file. The original file section looks like this:

```
# Normal logging
#log4j.appender.TRC.Threshold=OFF
#log4j.logger.com.google.gerrit.rpc=INFO
#log4j.logger.com.google=INFO
#log4j.logger.com.gerics=INFO
#log4j.logger.com.google.gerrit=INFO
#log4j.logger.com.google.gerrit.pgm.util.RuntimeShutdown$ShutdownCallback=INFO

# Verbose tracing (for troubleshooting purposes)
log4j.appender.TRC.Threshold=DEBUG
log4j.logger.com.google.gerrit.rpc=INFO
log4j.logger.com.google=DEBUG
log4j.logger.com.gerics=DEBUG
log4j.logger.com.google.gerrit=DEBUG
log4j.logger.com.google.gerrit.pgm.util.RuntimeShutdown$ShutdownCallback=INFO
```

Replace it with the following (uncomment normal logging and comment out the verbose tracing):

```
# Normal logging
log4j.appender.TRC.Threshold=OFF
log4j.logger.com.google.gerrit.rpc=INFO
log4j.logger.com.google=INFO
log4j.logger.com.gerics=INFO
log4j.logger.com.google.gerrit=INFO
log4j.logger.com.google.gerrit.pgm.util.RuntimeShutdown$ShutdownCallback=INFO

# Verbose tracing (for troubleshooting purposes)
#log4j.appender.TRC.Threshold=DEBUG
#log4j.logger.com.google.gerrit.rpc=INFO
#log4j.logger.com.google=DEBUG
#log4j.logger.com.gerics=DEBUG
#log4j.logger.com.google.gerrit=DEBUG
#log4j.logger.com.google.gerrit.pgm.util.RuntimeShutdown$ShutdownCallback=INFO
```

# TeamForge Git integration: History protection

History rewrites are non-fast-forward updates of remote `refs` and associated objects. History rewrites happen when a branch in a remote repository gets deleted, previously pushed commits get amended or filtered and forcefully re-pushed, or a remote branch/tag is pointed to an entirely different commit history.

History may get rewritten without leaving any trace of the previous state. Sometimes this behavior may be wanted — for example, in the case of removing code violating intellectual property, removing mistakenly committed large binary files or removing merged feature branches. The TeamForge Git integration therefore does not disable the history rewrite feature, but instead enables it for SCM Administrators alone. However, since rewriting history might be easily abused and result in accidental data loss, we've introduced the History Protection feature as a safety net and necessity for ensuring proper audit compliance.

History protection archives rewritten changes and keeps backups of deleted branches. If history changes occur, an immutable backup `ref` is created in the remote repository, notification emails are sent to all members of the Gerrit Administrators group, and an event is logged in the audit log. The backed up `ref` can be restored into a new branch with any Git client (without needing physical file access to the Gerrit server). Gerrit site administrators can still decide to remove selected backup `refs` permanently.

## TeamForge Git integration: Enable history protection

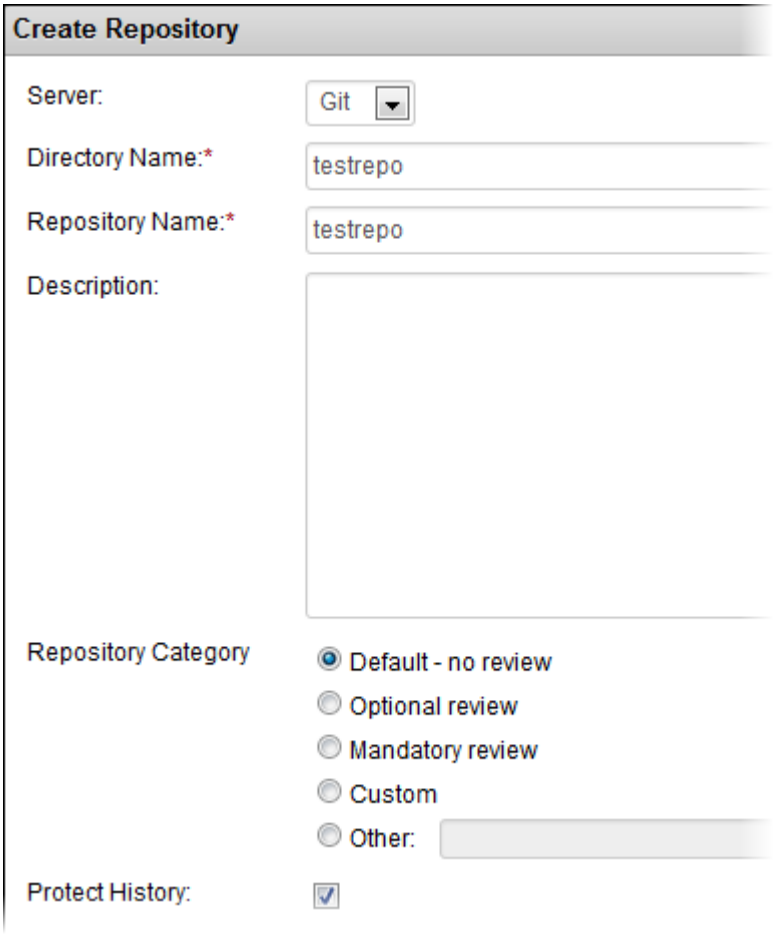
You can turn on history protection for the entire integration server or for an individual Git repository.

- To turn on history protection server-wide (for all the repositories hosted on the Git integration server), do the following:
  a) Set *`GERRIT_FORCE_HISTORY_PROTECTION`*`=true` in `/opt/collabnet/teamforge/runtime/conf/runtime-options.conf`.
  b) Restart the `gerrit` service.

```
/etc/init.d/collabnet restart gerrit
```

  **Note:** If you enable history protection server–wide, it cannot be overridden (turned off) for an individual Git repository. It will remain in effect for all Git repositories on the integration server.

- To turn on history protection for an individual Git repository in a TeamForge project, select the **Protect History** option while creating the repository. For an existing repository, do the following:
  a) On the **Source Code** page, select the Git repository and click **Edit**
  b) Select the **Protect History** option.




c) Click **Save**.

You can turn history protection on or off any time. However, your change will not be reflected in Gerrit immediately. It will be effective after the time that you defined as the regular refresh interval while installing the Git integration.

If you want your change to take effect immediately, do this right after you select or de-select the **Protect History** option: as a user with the Source Code Admin permission, temporarily remove any user having a project role with any SCM permission, and then add that user back. This will trigger an immediate sync which will enable history protection. After that, the Gerrit Administrator will be able to see History Protection enabled in the Gerrit web interface (by logging in as a Gerrit Administrator and clicking the General link for the project with the name of the Git repository).

**Note:** In TeamForge 6.2, history protection for a Git repository is turned on from the **Description** field. If you had turned on history protection in TeamForge 6.2, and have now upgraded TeamForge to version 7.1, you will see the **History Protect** option selected in the TeamForge 7.1 user interface. The **Description** field will still contain [Repo:ProtectHistory], but you can remove it since it does not have any effect in TeamForge 7.1.

**Related Links**

*TeamForge Git integration: History protection FAQ*

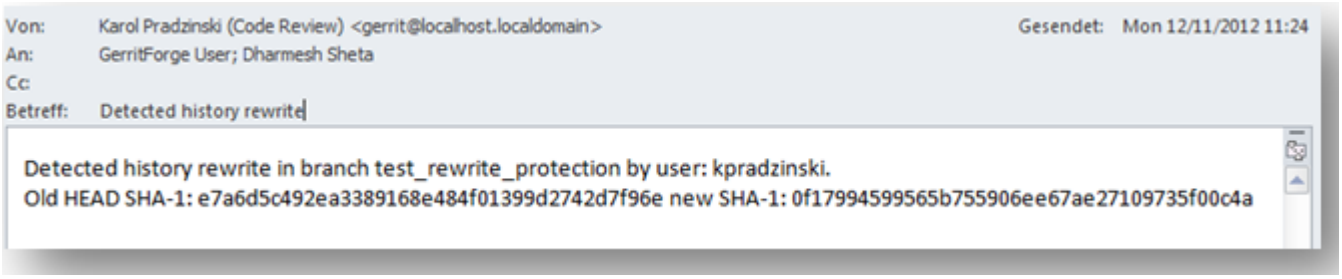
*History protection reports*

## History protection reports

Once history protection is turned on, any non-fast-forward push to a remote repository or deletion of a branch or tag on a remote repository is recorded and reported.

### Email notifications

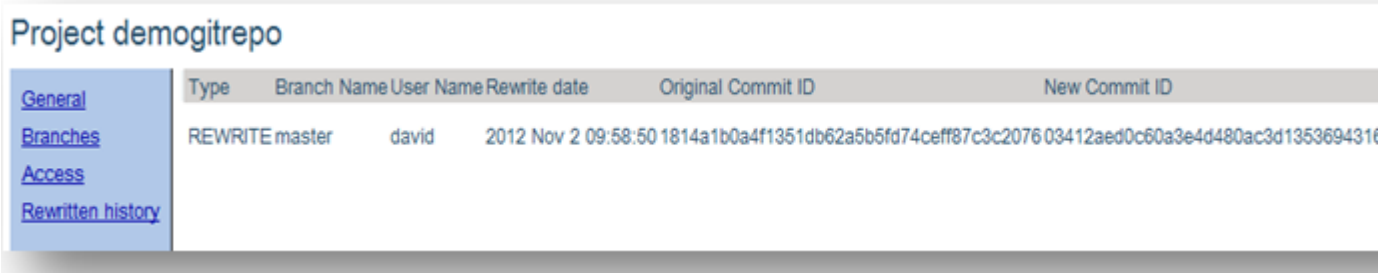
When history is rewritten, an email is sent to the Administrator group members in Gerrit.



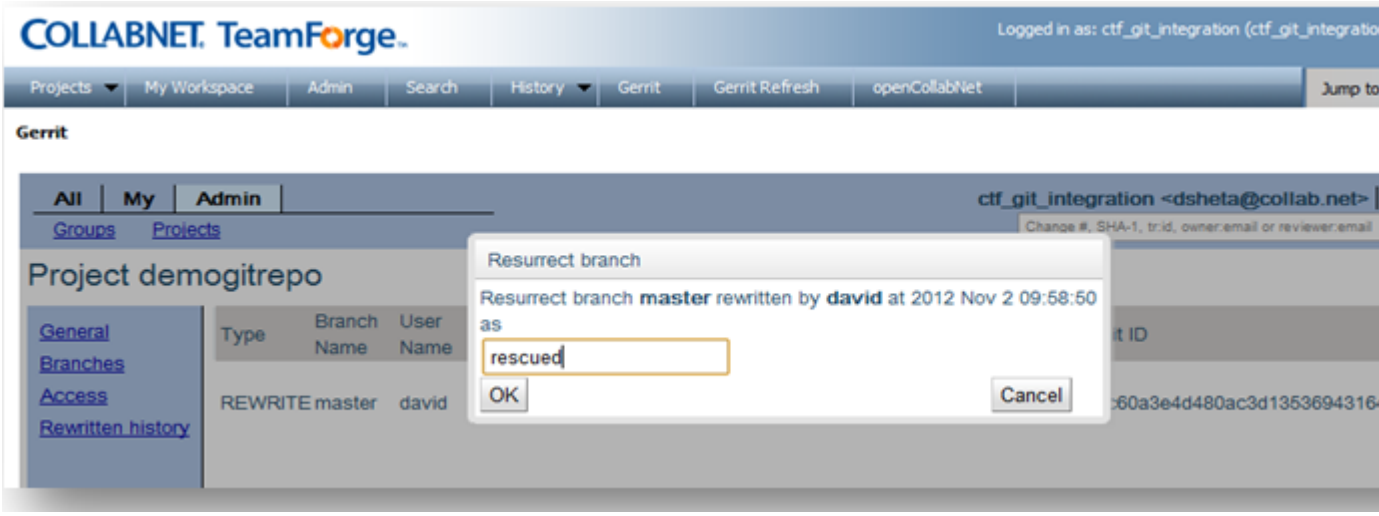

### Gerrit web interface

Every history rewrite event is logged and stored in the Gerrit database and visible in the Gerrit web interface. As Gerrit Administrator, you can —

- See rewritten history from **Project** > **Rewritten History**.



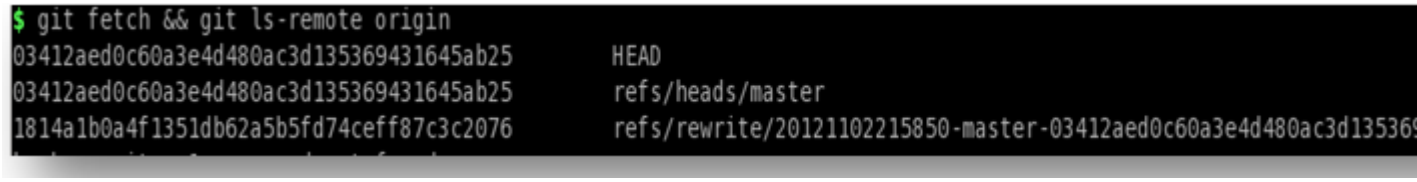

- Restore history by clicking **Resurrect** and providing a name for the new branch.




- Permanently remove a branch by clicking **Delete Permanently**.

### Git command line

You can use a standard Git client and run `git fetch && git ls-remote` for information on rewritten and deleted branches.




You can view entries in `refs/rewrite` (for non-fast-forward pushes) and `refs/delete` using the Git `ls-remote` command only if read access is granted to `refs/*`. Gerrit will prevent any other action such as delete/force-update on those special `refs` for all users including administrators.

### Audit log entries

The following events are logged in `/opt/collabnet/gerrit/logs/gerrit.audit.log`:

- Remote branches are deleted
- History is rewritten (non-fast-forward push)
- Backup branches are resurrected
- Backup branches are permanently deleted
- History Protection is turned on or off

### Related Links

*TeamForge Git integration: History protection FAQ*

*TeamForge Git integration: Enable history protection*

## TeamForge Git integration: History protection FAQ

Questions on history protection for Git repositories in TeamForge.

**What is history protection?**

History rewrites are non-fast-forward updates of remote `refs` and associated objects. History rewrites happen when a branch in a remote repository gets deleted, previously pushed commits get amended/tree filtered and forcefully re-pushed, or a remote branch/tag is pointed to an entirely different commit history. For more information, see *History Protection*.

**Is it possible to turn on history protection for all Git repositories hosted on a Git integration server? If yes, how?**

Yes, a user with file system access to the Unix machine where the Git integration is hosted can turn on history protection. First set the property `forceHistoryProtection = true` in `/opt/collabnet/gerrit/etc/gerrit.config` in the `[gerrit]` section. Then restart the `gerrit` service by running `$ service gerrit restart` on a shell.

With TeamForge 7.1, do the following:

- Set *`GERRIT_FORCE_HISTORY_PROTECTION`*`=true` in `/opt/collabnet/teamforge/runtime/conf/runtime-options.conf`.
- Restart the `gerrit` service.

```
/etc/init.d/collabnet restart gerrit
```

**I've enabled Protect History for my TeamForge project's Git repository. Will this be effective immediately?**

To enable history protection immediately, a TeamForge user with the Source Code Admin permission must do this right after enabling **Protect History**: temporarily remove any user with a project role with any SCM permission, and then add that user back. This will trigger an immediate sync after which history will be protected for the Git repository. Otherwise, history protection will be enabled after a periodical sync.

**Can I turn off history protection for any**

No, when history protection is enabled server-wide for the Git integration server, it cannot be turned off for a particular Git repository.

| | |
|---|---|
| **particular Git repository when it is enabled server-wide?** | |
| **Where can I see the backup branches generated by history protection?** | Backup branches are generated based on the type of History Rewrite. For a remote branch that is deleted, this is under `refs/delete`. For a non-fast-forward push, this is under `refs/rewrite` with the branch name containing the timestamp, original branch and the user who rewrote history, for example, `refs/delete/20121112042512-test--david`. |
| **Who can resurrect or permanently delete backup branches?** | A user who is a member of the Gerrit Administrator group can resurrect or permanently delete backup branches. By default, the TeamForge site administrator whose credentials are used for running the post-installation script is part of the Gerrit Administrator group. |
| **Who can see backup branches?** | By default, a TeamForge user with SCM View (or more) permission can see all backup branches by executing `git fetch && git ls-remote origin`. In Gerrit, the user must be part of a group which has at least read access for `refs/delete` and `refs/rewrite` for the given Gerrit project (TeamForge Git repository). |
| **Are the backup branches under `refs/rewrite` and `refs/delete` protected from Git garbage collection which removes unreferenced objects?** | Yes, objects in backup branches under `refs/rewrite` and `refs/deleted` are referenced and cannot be cleaned up by Git's garbage collection. |
| **Do backup branches take up a lot of disk space on the Git server?** | The backup branches on the Git server are mainly *Git objects* that are compressed deltas of original file versions. Git regularly compresses these objects to save disk space. |
| **What is the difference between Git *reflog* and History Protect?** | In Git, `reflog` records all activity on a branch, while History Protect only reports deleted branches/tags and history rewrites (non-fast-forward pushes) For details of all the differences, see *Git reflog vs TeamForge Git integration History Protect*. |

**Related Links**

*TeamForge Git integration: Enable history protection*

*History protection reports*

# GERRIT_FORCE_HISTORY_PROTECTION

The *GERRIT_FORCE_HISTORY_PROTECTION* token determines whether the history protection feature of the TeamForge Git integration is enabled. If it is set to TRUE, history protection is turned on for all repositories hosted on the Git integration server.

### Values

TRUE or FALSE

### Default

FALSE

### Comments

In TeamForge 7.0 (and later versions), this token is defined in the `runtime-option.conf` file to support the non-interactive installation of the Git integration. After Gerrit's post-install script is run, the value of this token is used to set the configuration property `forceHistoryProtection` in the `/opt/collabnet/gerrit/etc/gerrit.config` file.

For more information on enabling history protection, see *TeamForge Git integration: Enable history protection* .

# TeamForge Git integration: FAQ

Use this background information to set up, maintain, support and work with the TeamForge Git integration.

## TeamForge Git integration: Install FAQ

Questions about installing the TeamForge Git integration.

**What are the requirements for running the installer?**

For the required software, see *Requirements for the TeamForge Git integration*.

**Note:** In addition, you will need the following:

- `root`/`sudo` access to the machine
- A non-expiring, non-lockable TeamForge user account with site administrator permissions

**What does the installer consist of?**

The TeamForge Git integration is available as an add-on for TeamForge 6.2. The installer is an RPM package which you can run using a simple `yum` command. It contains a post-installation script which allows you to configure the integration to work with TeamForge.

**Do I need to shut down any TeamForge services to run the installer?**

No.

**Is there an alternative way to install TeamForge with the Git integration?**

Yes, CollabNet now provides *VMware images* for TeamForge with the Git integration pre-configured.

**How do I disable Gravatar support from the TeamForge Git Integration?**

If you have installed version 7.0.2 (or later) of CollabNet's customized Gitweb-caching, Gravatar support is enabled by default. To disable this support site-wide, comment out the following lines (by including a # at the beginning of each line) in the file `/var/www/gitweb-caching/gitweb_defaults.pl`:

```
'avatar' => {
                  'sub' => \&feature_avatar,
                  'override' => 0,
                  'default' => ['gravatar']},
```

**Related Links**

*Install the TeamForge Git integration*

## TeamForge Git integration: Post-install FAQ

Post-install questions on the TeamForge Git integration.

**Where can I locate the post-installation script?**

After the installation finishes successfully, you will find the post-installation script at `/usr/sbin/ctf-git-integration-setup.sh`.

**Where can I find logs for installation errors?**

The logs are located under `/tmp/ctf-git-integration-setup.log`. For more on log files, see the README and *TeamForge Git integration reference* .

**Why does the post-installation script ask for the TeamForge username and password? Is it safe to provide these values?**

The TeamForge user credentials you supply are used to synchronize TeamForge project roles and permissions. The credentials are encrypted -- so it is safe to provide them.

**How can I start and stop Git?**

After you successfully run the post-installation script, you can start (and stop) the Git service by running the following commands on a shell as the `root` or `sudo` user:

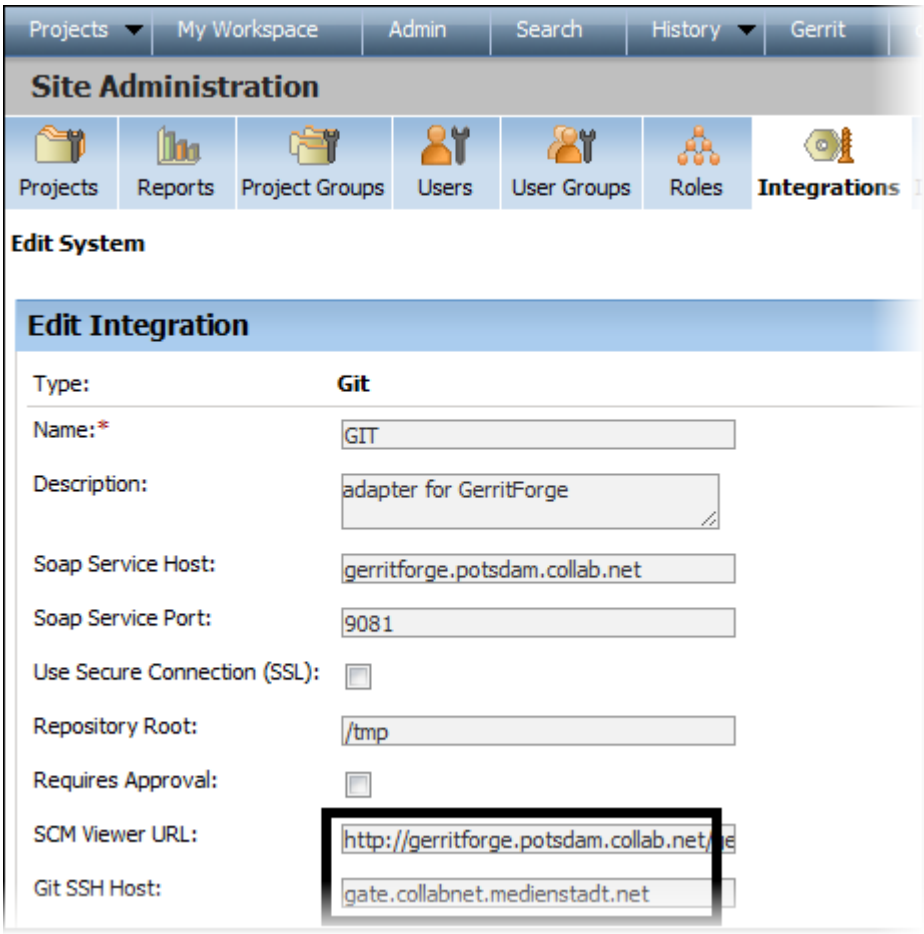
```
$ service gerrit start
$ service gerrit stop
```

**I provided some incorrect values while running the post-installation script. Can I change them?**

You can run the post-installation script `/usr/sbin/ctf-git-integration-setup.sh` at any point. The script will walk you through all configuration values, one by one, and ask if you want to change them.

**I'm using CollabNet's VMware image for TeamForge with the Git integration pre-configured. I want to change certain parameters of the Git SCM adapter, for example, the hostname of the server. How can I do that?**

CollabNet's VMware image has standard configurations for the Git integration. If you want to reconfigure a property (change the hostname from the default "localhost" to the FQDN), log in as the TeamForge site administrator and edit the property for the Git integration.




Make sure you retain the other configuration values.

# TeamForge Git integration: General usage FAQ

General usage questions about the TeamForge Git integration.

**What are the next steps after the post-installation script runs?**

After running the post-installation script, first create an SCM repository of type Git in your TeamForge project. Then add at least one project role with SCM permissions to access that repository and assign that role to one or more users. With the appropriate credentials (uploaded authorized keys) and the clone URL provided in the Source Code page, users will be able to clone the Git repository.

**I am a site admin/project admin/have global SCM permissions/looking at a public project. Why can't I access a newly created repository from the TeamForge web interface or clone it using my Git client?**

A new project role with source code permissions must exist. A user needs this role to access the repository from the TeamForge web interface. Project administrator rights, site administrator rights, global roles, and default access permissions for project membership are currently ignored by the Git integration.

**I've created a Git repository but it does not show up in Gerrit. Why might this happen?**

TeamForge repositories are synched only if there is at least one project role with SCM permissions in the corresponding TeamForge project. Once you create such a project role, the synch should happen, and the repository should appear as a project in Gerrit.

**How can I log into Gerrit?**

If your administrator has set up Gerrit as a linked application to TeamForge, you will automatically be logged into Gerrit (SSO) when you click its link. If not, access the URL `http(s)://<yourtfinstance>/scm integration server>/gerrit/` and provide your TeamForge credentials.

**What are the Git protocols that work with Git repositories managed by TeamForge?**

The Git integration currently allows you to access a Git repository using SSH. That said, you must have generated an SSH key pair and uploaded the SSH public key to TeamForge in **My Settings** > **Authorized keys** .

Alternatively, you can use http(s) to clone and push to Git repositories.

**Note:** Use this option only if SSH is not available to you.

To enable http(s) access, log into Gerrit and generate an HTTP password ( **Settings** > **HTTP Password** > **Generate Password** ). This password will not match your TeamForge password; you'll need to provide it to your Git client whenever you perform an operation that requires accessing the Git server. The clone URL for http(s) access follows this convention:

```
git clone https://$USERNAME@<yourtfinstance/scm integration
 server>/gerrit/p/<TFreponame>
```

**How do I generate an SSH key pair?**

You can generate an SSH key pair on a Unix machine by running the following shell command:

```
$ ssh-keygen -t rsa
```

(You will be asked to provide the location to store the key pair. The default is the home directory of the logged-in user.)

**After installing a Git client, I am able to clone a Git repository into my local work directory. However, I**

Right after you clone, but before you commit any changes locally, you will need to configure Git if you haven't already.

```
$ git config --global user.name "<TeamForge username>"
```

**am not able to "push" anything to the remote repository in spite of having view and commit permissions. What should I do ?**

```
$ git config --global user.email "<email used in TeamForge
for the user>"
```

You should now be able to push your changes.

**I've changed /appended my public key in TeamForge. Will I still be able to access a Git repository using the new SSH key pair?**

Yes, when you change your authorized key in TeamForge, it gets synced instantly with the Git integration. So you should be able to access you Git repository using a new key pair.

**Is a commit association created in TeamForge after I push my commit to a remote Git repository?**

Yes, when you push a local commit to the remote repository, an association will get created if the commit message contains a reference to a TeamForge item such as a tracker artifact, wiki or document.

**Note:** A commit association will not be created if you push your commit to Gerrit's "review branch" (push for review).

**What happens if the TeamForge site is down or there are some network problems -- will the Git integration still work?**

The Git integration will still work, but with the following limitations:

- If the TeamForge site is down, users will not be able to see commit associations created in TeamForge, but still be able to push commits to a Git repository.
- If the Git integration is hosted in LOCAL mode, network-related problems would definitely prevent changes being pushed to a Git repository.
- If the Git integration is hosted in REMOTE mode, the synchronization of roles and permissions will be cached during the period when TeamForge is down; Git will function with the roles and permissions synched already.

**What is a "Jumbo Push"?**

In contrast to Subversion, Git has the concept of local commits that stay in the local environment of a user, and at some point, get pushed to a remote repository all at once. This push checks in changes from all commits into the remote repository. For each of those commits, a commit object appears in the TeamForge (Source Code component). So, one push can have an unlimited number of commits and thus commit objects in TeamForge. You can, however, define the threshold for a single push based on how many commits should generate a commit object. A push' containing commits beyond that threshold is called a "Jumbo Push"'. You can configure the Jumbo Push threshold by running the post-installation script.

**What objects and relationships are mapped between TeamForge and the Git integration?**

See the README (APPENDIX, Relationship and Object mapping section) or *TeamForge Git integration reference* .

**When are the objects and relationships synchronized between TeamForge and the Git Integration?**

TeamForge project roles, project role SCM permissions, global groups, SCM repositories, and global group/project role membership are synched in two ways:

- Synchronously: after a regular interval (configurable using the post-installation script)
- Asynchronously: whenever there is a change related to roles or permissions within TeamForge, it triggers the sync between TeamForge and the Git integration.

TeamForge repositories are only synched if there is at least one project role with SCM permissions present in the corresponding TeamForge project.

TeamForge users are provisioned in Gerrit whenever you —

- Change their authorized keys in TeamForge

- Log into Gerrit by clicking the linked application link or using TeamForge username and password
- Access GitWeb (web interface for a Git repository) by clicking a Git repository link in the TeamForge Source Code page

**Note:** Changes in Gerrit are not synched back to TeamForge.

| | |
|---|---|
| **Where can I find system logs for the Git integration?** | You can find the logs under `/opt/collabnet/gerrit/logs/`. For more on log files, refer to the README or *TeamForge Git integration reference* . |
| **Can I bypass Gerrit and access a Git repository directly?** | No, Gerrit is used to enforce TeamForge access permissions. |
| **TeamForge supports an integration with Black Duck Code Sight. Does this work with Git?** | Yes. See *Set up Code Search for the TeamForge Git integration* for more information. |
| **I deleted a TeamForge Git SCM repository but the corresponding Gerrit project does not get deleted. What's wrong?** | Currently, Gerrit does not allow removing projects that are created already (so that you don't easily lose source code). One implication of this behavior is that even though you deleted the corresponding TeamForge repository, you will not be able to create a new one with the same directory name. |
| **How can I import an existing Git repository into Gerrit?** | As long as all the commits in the repository are yours and there is a linear history, a force push should be sufficient. Otherwise, you would have to go into Gerrit and manually add permissions *to push commits authored by other individuals* and *merge commits*. |
| **In the TeamForge web interface, I see the repository root parameter for Git set to "/tmp". Can I change that?** | For backward compatibility reasons, this parameter has to be set to "/tmp". It does not affect where Gerrit actually stores its Git repositories -- this is at `/gitroot`. |
| **Do we have default hook scripts available for Git in TeamForge?** | Associating artifacts based on commit messages and blocking commits without a commit message is a core TeamForge mechanism that is supported by Git as well.<br>To add hook scripts, see *http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/config-hooks.html*. |
| **Do we have email alerts for Git in TeamForge? If yes, where do we configure it?** | Email alerts based on TeamForge commits is a core TeamForge feature, independent of the SCM involved. In addition, Gerrit sends out review emails using the SMTP server specified during installation (it defaults to the TeamForge SMTP server). The mail template is explained in *http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/config-mail.html*. |
| **Do we have Role Based Access Control and Path Based Permissions for Git in TeamForge?** | We support all SCM permission cluster options for TeamForge project roles. Only TeamForge project roles are considered; default access permissions, global roles, project membership, site admin and project admin permissions are ignored. Path-based permissions are not relevant in Git since a Git commit always contains all files. If we did not ship certain files, this would result in a checksum error. Gerrit supports branch-based permissions but this feature is currently not directly exposed over the TeamForge web interface. |

# TeamForge Git integration: Upgrade and Uninstall FAQ

Questions on upgrading and uninstalling the TeamForge Git integration.

| | |
|---|---|
| **How do I upgrade the current version of my Git integration?** | You can upgrade by running the `yum update command`. See *Upgrade the Git integration along with TeamForge* for details. |
| **How can I update the configuration of the Git integration after upgrading?** | Run the post-installation script with the upgrade switch: `/usr/sbin/ctf-git-integration.sh --upgrade` |
| **Is there any downtime during the upgrade?** | Only the amount of time it will take to upgrade the binary and change the configuration (if required). |
| **How can I uninstall the Git integration?** | You can uninstall the integration by running the `yum remove` command. See *Uninstall the TeamForge Git integration* for details. |

**Related Links**

*Upgrade the Git integration along with TeamForge*

# TeamForge Git integration: Technical concepts FAQ

Questions on some of the TeamForge Git integration's more technical aspects.

| | |
|---|---|
| **How are TeamForge SCM permissions mapped to Gerrit Access Rights?** | The README describes how relationships (including permissions) are mapped between TeamForge and Gerrit -- see APPENDIX, relationship mapping or *TeamForge Git integration reference* for more information. The mapping from TeamForge SCM permissions to Gerrit Access Rights is defined in `/opt/collabnet/gerrit/etc/gerritforge.mappings`. While you can modify this file as you see fit, CollabNet officially supports only the default configuration. |
| **How can I fine tune access rights (read, write, review, submit) for certain users/groups in Gerrit?** | Have a look at the README and the help at *Work with Gerrit* . |
| **What does the directory structure look like for a Git integration?** | See the README or *TeamForge Git integration reference* . |
| **Where can I find more information about Gerrit?** | For more information on Gerrit, see the *Gerrit Community Documentation* page. |
| **Which ports does the Git Integration use? My organization has a strict firewall policy, and I need to know which ports to make available for the Git integration.** | The Git integration uses 3 ports: 9080,9081, and 29418. See the README or *TeamForge Git integration reference* for more information. For the integration, Git integration uses 3 ports(9080,9081,29418 follow details in README) Only port 29418 should be exposed by the firewall. |
| **I get a delay whenever I do a Git push, but not when I do a Git fetch. What is wrong?.** | When you do a Git push, Gerrit tries to do a reverse lookup of your IP address. If the nameserver configured for your Git integration server cannot do this reverse lookup, it will result in a timeout. You need to configure your nameserver list (`/etc/resolv.conf`) correctly. For further information, see *this core Java bug*. |
| **I ran Gerrit manually (without the service script; now my secure `config` file is gone and Gerrit does not start** | If Gerrit is run with a different Unix user than `gerrit`, newly created and modified files may not belong to the `gerrit` user any longer. As a consequence, when you try to restart Gerrit using its services script (which |

**up. What happened and how can I fix this?**

switches to the `gerrit` user), Gerrit might not start up due to wrong file permissions. If Gerrit detects that the permissions of its secure `config` file have been tampered with, it even removes this file. You should, therefore, only run Gerrit using the service script provide, and reconfigure it by running the post-install script again. You can fix incorrect permissions by running the following (as sudo or root):

```
chown -R gerrit.gerrit /opt/collabnet/gerrit
```

**I deleted the dedicated TeamForge Gerrit user account in TeamForge, and SCM permission synch is no longer working. How can I recover from this situation?**

- The easy, and recommended, approach is to ask CollabNet's Professional Services to undelete the TeamForge user in question.
- Otherwise, you would have to create a new dedicated site admin user in TeamForge, shut down Gerrit, re-run the post-install script and provide the credentials of that user. Then, you would have to start Gerrit again, and log in as that new user via the web interface. You will see that the user does not have any special admin permissions. If you still have a working user in Gerrit's administrator group, you could add the dedicated Gerrit user to that group using Gerrit's web interface. If not, you would have to manually add the new user to the Gerrit administrator group by shutting down Gerrit, removing all files from its caching directory, inserting the `user id` of the new user into Gerrit's Postgres `reviewdb` DB group/user membership table, and starting Gerrit again. Since this probably requires you to consult CollabNet's Professional Services as well, we strongly recommend the previous option (undeleting the previously removed user).

# TeamForge Git integration reference

Here's some reference information on Gerrit, integration with the CollabNet Desktops, and older releases..

## Gerrit directory structure, connectivity and more

Here's some reference information on Gerrit's scalability and hardware requirements, file/directory structure, database, ports and connectivity.

### Scalability and hardware requirements

- For estimated requirements and availability assumptions, see *http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/dev-design.html#_scalability*.
- For hardware considerations, see *http://code.google.com/p/gerrit/wiki/Scaling*.
- For a detailed description of performance-related settings, see *http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/config-gerrit.html*.

### Gerrit directory structure

Gerrit expects a standardized directory structure under the GERRIT_SITE directory: `/opt/collabnet/gerrit`, the `gerrit` user's home directory. The `/etc/default/gerritcodereview` file contains the GERRIT_SITE variable and points to `/opt/collabnet/gerrit`.

**Sub-directories of GERRIT_SITE** (`/opt/collabnet/gerrit`):

- GERRIT_SITE

- `.ssh`: contains the SSH key for the `gerrit` user; generated during installation and needs to be backed up.
- `bin`: binaries, startup script
- `gerrit.war`: the main Gerrit service
- `gerrit-sync.jar`: the Gerrit-TeamForge synchronization service
- `gerrit.sh`: SYSV-style init script; launches and shuts down; linked to `/etc/init.d/gerrit`
- `cache`: disk cache; does not need to be backed up; can always be re-generated on the fly; should be deleted after an upgrade or restore.

- `etc`: contains all configuration information; needs to be backed up.
- `gerrit.config`: the Gerrit configuration
- `secure.config`: obfuscated passwords and secrets
- `log4j.properties`: logging settings in log4j format
- `gerritforge.mappings`: defines how TeamForge access permissions are mapped to Gerrit access rights
- `lib`: libraries, potentially customer-specific extensions, treat like the `bin` directory

- `logs`: Gerrit log files; default configuration rotates logs daily, gzips old logs; debug files are rotated after they reach 10 MB; we keep 10 copies. You can make changes in Gerrit's `log4j.properties` file at `/opt/collabnet/gerrit/etc`.
- `audit.log`: audit events
- `system.log`: INFO-level logging
- `sshd.log`: logs connections to Gerrit's SSH port (not system shell SSH)
- `*.gc.log`: information on garbage collector usage

Note: In co-hosted mode, TeamForge log rotation behavior will be used as default.

- `/gitroot:` default location for Git repositories. The default location can be changed using the setting in `gerrit.config` or by symlinking the directory. You need to back up this directory.

### Database

Gerrit stores runtime information about users, groups, code reviews and commits in a PostgreSQL database called `reviewdb` by default. You need to back up this database. If the Git integration is installed on the TeamForge server, it will use the same PostgreSQL install as TeamForge. If the integration is on a separate server, it will use a local PostgreSQL installation on that server. During installation, a new `gerrit` role and `reviewdb` schema get created. Note that Git does not, at any point, access TeamForge schema within the Postgres database.

To access `reviewdb` from Gerrit, the following line will be added by the installer to `/var/lib/pgsql/9.0/pg_hba.conf`:

```
host     reviewdb          gerrit          samehost          md5
```

### Ports and connectivity

Gerrit opens three (TCP) ports: 9080, 9081 and 29418. You should expose only port 29418 outside localhost.

| | |
|---|---|
| **9080: the Gerrit Web interface** | This port will be proxied by Apache `conf` (prefix `/gerrit`) and doesn't need to be externally accessible. Do not expose this port to the outside. |
| **9081: Gerrit-sync web service (REST)** | In the Local (or co-hosted mode), TeamForge talks to the Gerrit REST API over localhost. Gerrit talks to TeamForge over its default SOAP URL.<br>The Git integration needs bidirectional connectivity to the TeamForge host. In the Remote (or distributed) mode, TeamForge talks to the Gerrit REST API over an Apache proxy rule (SSL-enabled if Apache is SSL-enabled on the integration server where Gerrit is running). Gerrit talks to TeamForge over its default SOAP URL.<br>Do not expose this port to the outside. |
| **29418: Gerrit SSH access** | Developers use the SSH protocol to push and pull source code to and from Gerrit. This port needs to be open to users.<br>**Note:** Gerrit ships its own SSH implementation and offers no shell access over this port. |

### Gerrit's integration with Continuous Integration (CI)

For an introduction on what the Gerrit review feature is about and how it works with CI, take a look at the following:

- *http://alblue.bandlem.com/Tag/gerrit/*
- *http://urlenco.de/vhqjl*
- Official Gerrit documentation on *http://source.android.com/source/life-of-a-patch.html*
- *http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/user-upload.html*

### Backward compatibility notes on `gerritforge.mappings`

If you are a current user and do not want the code review policy feature introduced in the TeamForge Git integration version 7.0.0, you can continue to use your `gerritforge.mappings` file without any behavioral changes. To determine whether you are still using the old `gerritforge.mapping` file format, there is a new property called `mapping_format_version`. Its default value (if not mentioned) is 1. In the new template file, it is set to 2. This allows for future changes to the format. If the number cannot be parsed, we issue an error message and assume 2.

The following two paragraphs are very technical — they may not interest you if you want to upgrade, unless you've made extensive manual changes to access rights directly in the Gerrit web interface or use the programmatic access right infrastructure.

In the backward compatibility mode (`mapping_format_version`codeph> < 2), the only managed categories are READ, pHD, pTag, and OWN. Access rights of other categories (such as submit, review, forge identity) are be touched by GerritSynch. In `mapping_format_version`codeph> == 2, we support all categories in Gerrit 2.1.8 and if `[<name of Git Repo Category>.]keep_rights_added_in_gerrit` is set to false (which is the case

for the default category, mandatory review and optional review), we wipe out access rights of previously unmanaged categories as well.

If a Gerrit access right category code is mentioned for a repository category, all previously existing access rights of that access right category are replaced as long as `[<name of Git Repo Category>.]keep_rights_added_in_gerrit` is set to false. If this property is set to true, and the backward compatibility mode is turned on, we only keep the existing access rights if they do not refer to `refs/*` or `refs/tags/*`. If this property is set to true and `mapping_format_version`codeph> is set to 2, we keep the existing access rights no matter what they refer to.

### Mapping

Objects and Relationships are mapped from TeamForge to Gerrit, not the other way. The mapping rules are defined in `/opt/collabnet/gerrit/etc/gerritforge.mappings`.

| TeamForge Object | Gerrit Object |
|---|---|
| SCM repository in TeamForge project (containing project roles with SCM permissions) | Project |
| Project Role | Group |
| User Group | Group |
| Project Role with SCM permission | Access right |
| User | User |

| TeamForge Relationship | Gerrit Relationship |
|---|---|
| SCM repository in TeamForge project with project roles with SCM permissions | SCM repository in TeamForge project with project roles with SCM permissions |
| User is part of a User Group that is assigned a Project Role | User is part of a Group (which corresponds to a TeamForge Project Role) |
| User is part of a User Group | User is part of a Group (which corresponds to a TeamForge User Group |
| Project Role is assigned SCM Admin permission (such as Admin, Delete and View, View and Commit, View Only, None) | Corresponding group is assigned Gerrit access rights matching the assigned TeamForge SCM permissions. The access rights are determined by the code review policy of the corresponding TeamForge repository |

**Note:** In this release, project administrator rights, site administrator rights, global roles,and default access permissions for project membership are ignored.

**Note:** For the corresponding mappings in the first version of the TeamForge Git integration (version 6.2), see *this table*.

| Gerrit Category Code | Gerrit Access Rights Category |
|---|---|
| READ | Read Access |
| pHD | Push Branch |
| pTAG | Push Tag |
| OWN | Owner |
| CRVW | Code Review |
| FORG | Forge Identity |

| Gerrit Category Code | Gerrit Access Rights Category |
|---|---|
| SUBM | Submit |
| VRIF | Verified |

For more information about access control categories and their ranges in Gerrit, see *http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/access-control.html*.

# TeamForge Git integration with the CollabNet Desktops

The CollabNet Desktops for Eclipse and Microsoft Visual Studio integrate smoothly with the TeamForge Git integration.

When you browse TeamForge's Git repositories, you will be able to clone them directly from your IDE. Eclipse will also detect Gerrit and expose its reviews in the task list.

If you have configured the Jenkins trigger plugin and Jenkins comments on the uploaded change sets, you will be able to directly navigate to the build in question. For more information, see *http://tasktop.com/blog/eclipse/stage-build-review-with-git-gerrit-hudson-and-mylyn*.

# TeamForge Git integration archives

This section contains legacy documentation related to earlier versions of the TeamForge Git integration.

## Mappings between TeamForge and Gerrit

This table shows how objects and relationships are mapped between TeamForge and Gerrit in the first version of the TeamForge Git integration (version 6.2).

| TeamForge Relationship | Gerrit Relationship |
|---|---|
| SCM repository in TeamForge project with project roles with SCM permissions | SCM repository in TeamForge project with project roles with SCM permissions |
| User is part of a User Group that is assigned a Project Role | User is part of a Group (which corresponds to a TeamForge Project Role) |
| User is part of a User Group | User is part of a Group (which corresponds to a TeamForge User Group |
| Project Role is assigned SCM Admin permission | Corresponding group is assigned Gerrit access rights (category code, lower range, upper range):<br><br>READ,1,2<br>pHD,1,3<br>pHD,1,3,refs/tags/*<br>pTAG,1,2<br>OWN,1,1 |
| Project Role is assigned SCM Delete permission | Corresponding group is assigned these Gerrit access rights (category code, lower range, upper range):<br><br>READ,1,2<br>pHD,1,3<br>pTAG,1,2 |

| **TeamForge Relationship** | **Gerrit Relationship** |
|---|---|
| Project Role is assigned SCM View and Commit permission | Corresponding group is assigned these Gerrit access rights (category code, lower range, upper range):<br><br>READ,1,2<br>pHD,1,2 |
| Project Role is assigned SCM View Only permission | Corresponding group is assigned these Gerrit access rights (category code, lower range, upper range):<br><br>READ,1,1 |
| Project Role is assigned SCM No Access permission | Corresponding group is assigned these Gerrit access rights (category code, lower range, upper range):<br><br>READ,-1,-1 |
| Registered user in TeamForge who has logged into Gerrit at least once | Implicit access rights in all Gerrit projects (category code, lower range, upper range):<br><br>CRVW, -1, 1<br>FORG, 1, 1 |

# TeamForge Git integration release notes

Look here for information about releases of the Git integration supported by TeamForge 7.1.

## TeamForge Git integration 7.1.4 release notes

Release date: December 2013

### Fixed issues: TeamForge Git integration 7.1.4

Version 7.1.4 of the TeamForge Git integration includes this fix.

| |
|---|
| Mylyn Gerrit connectors for Eclipse can now parse the version string of the TeamForge Git integration. |

## TeamForge Git integration 7.1.3 release notes

Release date: July 2013

### Fixed issues: TeamForge Git integration 7.1.3

Version 7.1.3 of the TeamForge Git integration includes these improvements and fixes.

| |
|---|
| This release includes code changes for handling the special TeamForge Code Search user account. |
| A Null Pointer Exception that was possible when the repository description is `null` is fixed. |

## TeamForge Git integration 7.1.2 release notes

Release date: July 2013

### Fixed issues: TeamForge Git integration 7.1.2

Version 7.1.2 of the TeamForge Git integration includes the fix for a JGit security issue that does not enforce branch-based read-access rights correctly.

The original security issue was reported here: *https://groups.google.com/forum/#!topic/repo-discuss/DaR64jwRhpI*.

For most users of the TeamForge Git integration, this is unlikely to pose a problem because someone exploiting the security vulnerability would need to know or have the following:

- The SHA-1 of a commit for which they do not have access (because the commit was made in a branch for which they do not have access) — SHA-1 are close to impossible to guess; so somebody would have to provide them that information or they would have to get it from a mailing list or bug tracker.
- Legitimate read-access to at least one branch in the repository
- Path-based permissions — this is not a common scenario since the built-in TeamForge repository categories of default review, optional review and mandatory review either grant access to all branches or no branch, so read-access to at least one branch in the repository (previous point) is not provided
- How to exploit the bug and modify your Git client accordingly — Google has fixed the problem and announced the fix publicly mentioning only that the problem is not exploitable with an ordinary Git client

| |
|---|
| For more information, see<br>*http://blogs.collab.net/teamforge/security-fix-for-gerrit-please-update-your-teamforge-git-integration* |

# TeamForge Git integration 7.1.0 release notes

Release date: July 2013

## New features: TeamForge Git integration 7.1.0

Version 7.1.0 of the TeamForge Git integration adds these new features.

### Highlights

| |
|---|
| This release supports *TeamForge 7.0*. |
| There is a *new, non-interactive process for installation* with TeamForge 7.0. |
| With TeamForge 7.0, this release is supported on SuSE. |

## Fixed issues: TeamForge Git integration 7.1.0

Version 7.1.0 of the TeamForge Git integration includes these improvements and fixes.

| |
|---|
| artf146083: The local timezone was displayed instead of GMT in `gitweb-caching`. |
| artf144815: Go-URLs generated by the linkification feature in GitWeb did not convert all upper case characters to lower case. This resulted in broken links. |
| Gravatar images displayed in `gitweb-caching` are now fetched over a secure HTTP connection. |
| The JGit library has been updated so that a vulnerability detected in earlier versions is fixed. |