CollabNet TeamForge Git Integration README

TeamForge (TF) now supports Git- a powerful distributed version control system. CollabNet's Git integration is powered by Gerrit, an open source Git server which enforces access control on Git repositories and supports powerful review features and integration points for continuous integration tools like Jenkins.

In nutshell TeamForge Git Integration features are:

- 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
- TF Artifact association with Git commits
- Git source code browsing and code search
- SSO between TeamForge and Gerrit's Web UI
- Full support of Gerrit's APIs to connect with CI systems like Jenkins

Installation Scenario Overview

Our Git Integration with TeamForge supports the following installation modes:

1. Local/cohosted mode: TeamForge and the Git integration server are running on the same machine.

Note: Local TF Git Integration is only supported if TeamForge is configured with Postgres DB running on the same machine, NOT supported when TeamForge is using Oracle DB or Postgres DB running on a different box. If those conditions do not apply for your environment, please use one of the remote mode variants.

2. REMOTE/distributed mode: TeamForge and the Git integration server are hosted on different machines

Variant 1): Git integration is hosted on an TF SCM integration server like remote CVS and SVN integrations

Variant 2): "Free form mode": Git Integration is hosted on an ordinary RHEL machine, manual steps are probably required to complete the installation since environments vary too much to automate every install steps.

It is possible to setup multiple Git integration servers, but only one can be installed in cohosted mode.

Requirements

- Operating System: RHEL / CentOS: 5.6 or later
- TeamForge 6.2
- JRE: 1.6 or later (Oracle JRE only)
- git: 1.7 or later
- postgresql-server : 9.0.x and permissions for gerrit Unix user to connect to DB reviewdb from localhost
- postgresql(client)
- oppenssh-clients
- gitweb-caching
- Apache Webserver with proxy and rewrite modules enabled
- TeamForge host SSL certificate imported into Git Integration server's JVM trust store
- Gerrit server host certificate imported into TeamForge server's JVM trust store

If you are installing in local mode or on an existing SCM integration server, those packages will be part of CollabNet's yum repositories and the required modifications on Apache's and Postgresql's configuration will be automatically done by the installer.

In "free form mode" (variant 2, remote mode), you would have to make sure to install those packages yourself and modify your Apache/Postgresql configuration yourself.

NOTE: If you use installation scenarios 1 or 2 variant 1, CollabNet has already uploaded the required RPMs in the configured yum repositories, in free form mode, you may have to configure additional yum repositories.
For RHEL 6 and later, Gitweb-Caching is part of the native OS binary. If not, add EPEL repository as follows:
(32-bit) $ rpm –ivh http://download.fedoraproject.org/pub/epel/6/i386/epel-release-6-7.noarch.rpm
(64-bit) $ rpm -ivh http://download.fedoraproject.org/pub/epel/6/x86_64/epel-release-6-7.noarch.rpm
--For RHEL 5.6, add the EPEL repository as follows:
(32-bit) $ rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/i386/epel-release-5-4.noarch.rpm
(64-bit) $ rpm -Uvh http://dl.fedoraproject.org/pub/epel/5/x86_64/epel-release-5-4.noarch.rpm

The installer always requires a dedicated, TeamForge site admin account that does not expire (see USERS_WITH_NO_EXPIRY_PASSWORD in TF site options).

TeamForge also has to be configured to allow creating SCM integration servers (DISABLE_CREATE_INTEGRATION_SERVER in TF site options has to be set to false [default value])

Install, Upgrade and Uninstall Instructions

The TeamForge Git integration is provided in an RPM package called `ctf-git-integration.rpm`

You have either received this RPM as a separate download or as part of a CollabNet yum repository.

Install

Perform the installation on the machine where the Git integration server is to be hosted.

Do the following **while logged in as root or using sudo**:

1 a) If you have downloaded the RPM separately
Run `yum install ctf-git-integration.rpm  --nogpgcheck`

1 b) If you are using CollabNet's yum repository
Run `yum install ctf-git-integration --nogpgcheck`

2) Follow the instructions at the end of the installation process and run the post install setup script specified: `/usr/sbin/ctf-git-integration-setup.sh`

NOTE: The post installation script will do its very best to auto detect all required configuration parameters to auto-configure the Git integration correctly. It is safe to accept the default values unless you explicitly want to override them. There are only three parameters without default value:

- TF user name: This is the site admin account mentioned in the prerequisites
- TF password: This is the password of the site admin account mentioned in the prerequisites
- Gerrit DB password: This is the password to protect Gerrit's DB from unauthorized access, you can decide upon its value when you first run the post install script. Its value is questioned later in the same script again, so please note it down properly. Do not enter an empty password here.
- If you are running the installer in freeform mode (REMOTE MODE, variant 2), you will need to provide values for two additional settings:

- TeamForge *hostname:* This should be the externally visible host name end users type into their browser if they want to reach TeamForge, so http://localhost is properly not a good idea.

- *SCM Shared secret key*: You can retrieve this value from `/opt/collabnet/teamforge/runtime/conf/runtime-options.conf` on your TF host

The post-installation script will automatically create a Git SCM adaptor in TF and set it up properly. You can tweak the configuration settings in TeamForge's Integration section if necessary. Please do not change the value of the repository root parameter *("/tmp")* though, 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 installation script has been completed successfully, please start the gerrit service (see section Starting and stopping Gerrit) and check `/opt/collabnet/gerrit/logs/gerrit-synch.system_log`,and `gerrit.system_log` for any error messages (see APPENDIX for further information on where to find log files).

If the log indicates no error messages, you have successfully setup the Git integration. Please refer to the manual post installation section for the last actions necessary to enable CodeSearch for Git and access the Gerrit Web UI directly from TF. If you see error messages in the logs, you can always reconfigure Gerrit (see Re-Configure).

Upgrade

Do the following while *logged in as root or using sudo*:

1 a) If you have downloaded the RPM separately
Run `yum update ctf-git-integration.rpm --nogpgcheck`

1 b) If you are using CollabNet's yum repository
Run `yum update ctf-git-integration --nogpgcheck`

2. To upgrade the configuration, run
`/usr/sbin/ctf-git-integration-setup.sh --upgrade`

Remove all files under `/opt/collabnet/gerrit/cache` before starting the Gerrit service again.

If you upgraded from TF Git Integration 6.2 to a newer version, the integration will behave exactly as before the upgrade as far as access right mappings are concerned (see appendix Legacy Documentation). If you want to take advantage of the new SCM code review policy feature, you would have to delete `/opt/collabnet/gerrit/etc/gerritforge.mappings` and restart Gerrit. Before doing this step, please carefully read appendix " *gerritforge.mappings Backward Compatibility Notes*".

Uninstall

Run as root or sudo:
`yum remove ctf-git-integration`
This will not remove your git repositories from the hard disk, see appendix if you want to remove those as well.

Re-configure

Please stop the gerrit service first before making any modifications on `/opt/collabnet/gerrit/etc/gerrit.config`. Then run *(sudo or root)*
`/usr/sbin/ctf-git-integration-setup.sh`
Then, start the Gerrit service again.

Post Installation Steps

Starting and Stopping Gerrit

Git(Gerrit) can be started, stopped, restarted and checked by running following command *as root or sudo* on a shell
`service gerrit start`

```
service gerrit stop
service gerrit restart
service gerrit check
```

Adding Gerrit as Linked app

It is advised to add Gerrit as a TeamForge site wide and project wide linked app so it can be accessed easily from TeamForge and does not require to relogin again. Prepare a link http://<GERRITHOSTNAME>/gerrit/sso/ (last forward slash really matters here, please do not omit it)

- Login to TeamForge as a site admin
- Goto *Admin Tab--> click Integrations-->Site-Wide Linked Applications*
- Create an app by giving it a meaningful name (like Gerrit, or something more descriptive if you have multiple Git integrations), paste the link prepared and choose SSO on and IFrame and in the end hit save button.
- You will see site linked app created

Similarly for Project wide linked app, go to Project Admin-->Project Toolbar --> Linked Applications. Create a new one by giving it a meaningful name (e. g. Gerrit), use the link prepared above as target, select an icon and save.

Setting up CodeSearch for the Git Integration

If you want to use TF CodeSearch for Git, you have to manually grant the TF CodeSight user “scmviewer” permissions to access all Git repositories. A prerequisite for this is to import CodeSearches’s public key into TeamForge. This step is described in the TeamForge CodeSearch documentation. Once this is done, you have to mirror the TeamForge CodeSearch user account “scmviewer” to gerrit by running the following command in a shell on the host where you have installed the GIT integration:

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

Once the scmviewer user is present in Gerrit, you have to grant it read permissions for all Gerrit projects by logging into Gerrit’s Web UI as a Gerrit super user (See appendix on Gerrit Access Rights Fine Tuning):

- create an internal Gerrit group (CodeSight group)
- add the scmviewer to the group
- Select Category as 'Read Access'
- Enter the group name you have created under Group Name field.
- Enter Reference name field as "refs/*"
- Permitted Rage as +1:Read Access
- Click the button "Add Access Right"
- Logout from Gerrit

After initial reindexing, you should now be able to use CodeSearch with Git.

Further documentation

The latest help on the Git Integration can be found under http://help.collab.net

APPENDIX

TF objects and relationships mapped to Gerrit

TeamForge Object	**Gerrit Object**
SCM Repository in TF project with project roles with SCM permissions	*Project*
Project Role	*Group*
User Group	Group
Project Role SCM Permission	Access Rights
User	User

TF Relationship	**Gerrit Relationship (also see table with category codes)**
SCM Repository in TF project with project roles with SCM permissions	*SCM Repository in TF project with project roles with SCM permissions*
User is part of a User Group which is associated with a Project Role	*User is part of Group(which corresponds to TF Project Role)*
User is a part of a User Group	*User is part of Group(which corresponds to TF User Group)*
Project Role is assigned an SCM Permission (like Admin, Delete&View, View and Commit, View Only, None)	*Corresponding group is assigned Gerrit access rights matching the assigned TF SCM permissions. Which access rights those are is determined by the code review policy of the corresponding TF repository (see next section).*

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

Project admin rights, site admin rights, global roles, project groups, project group roles and default access permissions, project membership are ignored.

Gerrit Category Code	**Gerrit Access Rights Category**
READ	**Read Access**
pHD	**Push Branch**
pTAG	**Push Tag**
OWN	**Owner**
CRVW	**Code Review**
FORG	**Forge Identity**
SUBM	**Submit**
VRIF	**Verified**

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

Control Code Review Policy per Git Repository

Repository specific TF permissions - Gerrit access rights mapping

For **TeamForge Git Integration versions greater than 6.2**, users can control all Gerrit Code Review features directly out of TeamForge. This is done by specifying a code review policy as part of the TF Git repository description field. By default, we ship the following code review policies:

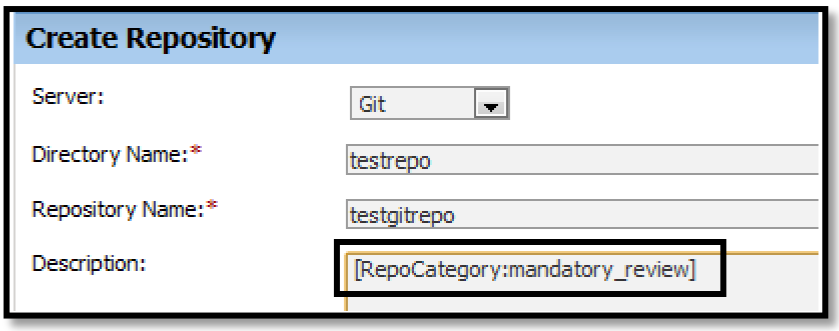
- Default (no special code review policy selected): All Gerrit review features are turned off, read/write access rights are enforced
- Mandatory Review: All code changes have to be reviewed, read/write access is enforced
- Optional Review: Review feature is turned on but can be bypassed if necessary, read/write access is enforced
- Custom: Access rights have to be set manually in Gerrit Web UI and will not be overridden by TeamForge. This is for advanced groups who are deeply familiar with Gerrit acces rights and want to turn off "auto pilot".

- Customer having access to gerritforge.mappings file may also introduce their own categories, see following sections for details.

Customers upgrading from **TeamForge Git Integration v 6.2** should carefully read our upgrade section to find out how to use this feature and how this will change default access right mappings.

Mandatory review:

If keyword `[RepoCategory:mandatory_review]` is specified in description field of a TF Git repository then every single commit pushed to this repository has to pass through review process before getting committed (merged) to this Git repository. Only TeamForge user with **source code admin** permission can bypass review.



Create Repository

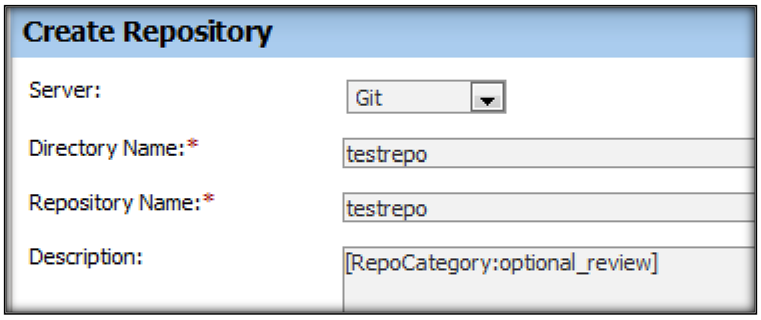
Server:	Git
Directory Name:*	testrepo
Repository Name:*	testgitrepo
Description:	[RepoCategory:mandatory_review]

- Users with no permissions(No access) cannot do anything

- Users with read permissions(View only) can read branches and push for review, have -1 and +1 for reviews
- Users with commit permissions(Commit/view) can do everything read permissions would grant + -2, +2 for reviews, verify and submit permissions (no right to bypass review)
- Users with delete permissions (Delete/View)can do everything commit permissions would grant
- Users with admin permissions can do everything delete permissions would grant + push to/create any branch (bypassing review), rewrite history, forge identity of Gerrit server, right to push tags, right to upload merges, right to fine tune access rights in Gerrit for the Gerrit project in question

Optional Review

*If keyword [RepoCategory:optional_review] is specified in description field of a TF Git repository, every single changes submitted to Git Repository' in TeamForge can be pushed for code review or it can be pushed to repository directly, bypassing review, when TF user has role with appropriate permissions (**source code delete/view and commit view** permission - or –TeamForge user with **source code admin** permission).*



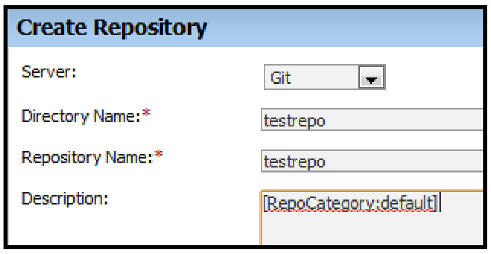
Create Repository

Server:	Git
Directory Name:*	testrepo
Repository Name:*	testrepo
Description:	[RepoCategory:optional_review]

- Users with no permissions (No access) cannot do anything
- Users with read permissions(View Only) can read and push for review, have -1 and +1 for reviews
- Users with commit permissions(Commit/View) can do everything read permissions would grant + -2, +2 for reviews, verify and submit permissions, push to/create any branch (bypassing review) and pushing tags
- Users with delete permissions(Delete+View) can do everything commit permissions would grant + right to rewrite history, upload merges and forge identity
- Users with admin permissions can do everything delete permissions would grant + forge identity of Gerrit server +
- right to fine tune access rights in Gerrit for the Gerrit project in question

Default Category

If keyword `[RepoCategory:default]` *is specified in description field or no keyword is specified at all, default code review policy is assumed which only enforces read/write access rights and do not expose Gerrit's review features for the corresponding repository.*



Create Repository

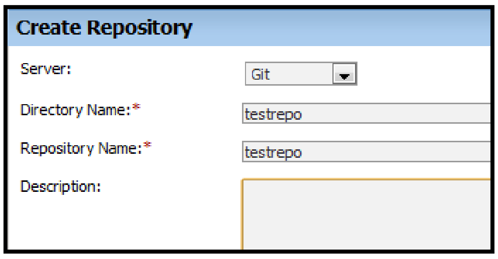
Server: Git

Directory Name:* testrepo

Repository Name:* testrepo

Description: [RepoCategory:default]

-or-



Create Repository

Server: Git

Directory Name:* testrepo

Repository Name:* testrepo

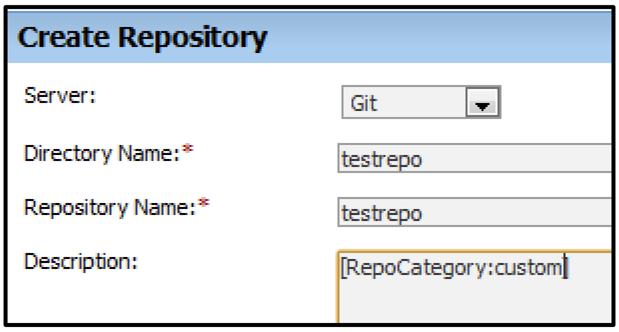
Description:

- Users with no permissions(No access) cannot do anything
- Users with read permissions(View Only) can only read branches
- Users with commit permissions (Commit/View)can do everything read permissions would grant + push to/create any branch (bypassing review), and push tags, review features are not turned on
- Users with delete permissions (Delete/View) can do everything commit permissions would grant + right to rewrite history + upload
- merges + forge identity
- Users with admin permissions can do everything delete permissions would grant + forge identity of Gerrit server +
- right to fine tune access rights in Gerrit for the Gerrit project in question

Custom Category

If keyword `[RepoCategory:custom]` is specified in the description field of a TF Git repository, users with TeamForge role with source code admin permission are able to fine tune permissions (access rights) in Gerrit's Web UI directly. Those changes will not be overridden by TeamForge.

Please refer to section "Gerrit Access Rights Fine Tuning"to see how to manually define access rights in Gerrit Web UI.



Create Repository

Server: Git

Directory Name:* testrepo

Repository Name:* testrepo

Description: [RepoCategory:custom]

- Users with no permissions(No Access) cannot do anything, unless added in Gerrit (will not be overwritten by TF)
- Users with read permissions(View only) cannot do anything, unless added in Gerrit (will not be overwritten by TF)
- Users with commit permissions(Commit/View) cannot do anything, unless added in Gerrit (will not be overwritten by TF)
- Users with delete permissions (Delete/View)cannot do anything, unless added in Gerrit (will not be overwritten by TF)
- Users with admin permissions have right to fine tune permissions in Gerrit for the Gerrit project in question

In case you decide to change repository description from some other code review policy to [RepoCategory:custom] the currently set access rights will not be touched by the synchronization. Subsequent changes to SCM permissions assigned to prject roles of the corresponding TF project will have no impact in Gerrit. The custom category is basically turning off “auto pilot”.

Overall Mapping TeamForge Source Code Permissions to Gerrit Access Rights per 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`
			`pHD ,1,2`
			`pTAG,1,2,refs/tags/*`
				`READ,1,3`
				`pHD ,1,3`
				`pTAG,1,2,refs/tags/*`
				`FORG,1,2`
					`READ,1,3`
					`pHD ,1,3`
					`pTAG,1,2,refs/tags/*`
					`FORG,1,3`
					`OWN ,1,1`

mandatory _review	`READ,-1,-1`	`READ,1,2`
		`CRVW,-1,1`
			`READ,1,2`
			`CRVW,-2,+2`
			`VRIF,-1,1`
			`SUBM,1,1`
				`READ,1,2`
				`CRVW,-2,+2`
				`VRIF,-1,1`
				`SUBM,1,1`
					`READ,1,3`
					`CRVW,-2,+2`
					`VRIF,-1,1`
					`SUBM,1,1`
					`pHD ,1,3`
					`pTAG,1,2,refs/tags/*`
					`FORG,1,3`
					`OWN ,1,1`

optional _review	`READ,-1,-1`	`READ,1,2`
		`CRVW,-1,1`
			`READ,1,2`
			`CRVW,-2,+2`
			`VRIF,-1,1`
			`SUBM,1,1`
			`pHD ,1,2`
			`pTAG,1,2,refs/tags/*`
				`READ,1,3`
				`CRVW,-2,+2`
				`VRIF,-1,1`
				`SUBM,1,1`
				`pHD ,1,3`
				`pTAG,1,2,refs/tags/*`
				`FORG,1,2`
					`READ,1,3`
					`CRVW,-2,+2`
					`VRIF,-1,1`
					`SUBM,1,1`
					`pHD ,1,3`
					`pTAG,1,2,refs/tags/*`
					`FORG,1,3`
					`OWN ,1,1`

custom	-	-	-	-	`OWN ,1,1`

Defining your own Code Review Policy/Modifying existing Code Review Policies

In case organizations want to customize existing code review policies or tailor their own, they can do so by modifying `/opt/collabnet/gerrit/etc/gerritforge.mappings` (which requires file system access to the Git Integration Server). It is advised to rather create a new code review policy than modifying the existing ones as long as you are still experimenting with the exact settings. If you delete `gerritforge.mappings` from the file system, it will be automatically recreated with the factory defaults after Gerrit restart.

gerritforge.mappings is a Java property file with format

```
[<name of Git Repo Category>.]<TF 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 TF 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 TF project role are mentioned in the mapping rules file for the corresponding repository category, none will be assumed. If none is not mentioned in the mapping rules, no access right are 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 `[<name of Git Repo Category>.]keep_rights_added_in_gerrit` is set to false. If this property is set to true, existing rights will be kept.

`[<name of Git Repo Category>.]keep_rights_added_in_gerrit` also determines, whether access rights of Gerrit categories not explicitly mentioned for the repository category in question will be deleted or kept.

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

The default code review policies can be expressed as

1) default category: `[RepoCategory:default]` or no mention in repo description at all

2) mandatory review: `[RepoCategory:mandatory_review]`

3) optional review: `[RepoCategory:optional_review]`

4) custom: `[RepoCategory:custom]`

Gerrit Access Rights Fine Tuning

Manually adjusting Git repository default access permissions in Gerrit

By default, Gerrit projects (TF Git repositories) are only visible to TF users which are assigned to a project role with SCM permissions. If you want to grant additional permissions to all registered users (like read access), to TF global groups or a custom subset of users (Gerrit internal group), you would have to log directly into Gerrit (http(s)://<GERRITSERVERHOSTNAME>/gerrit/) or click on the linked application button. Gerrit default access permissions can only be changed by Gerrit super users. By default, the dedicated TF user you provided during the install process is the only super user. Once you have logged into Gerrit's Web UI with that super user, click *on 'Admin' tab and navigate to Projects---> -- All Projects -- --> Access*

From here, you can control access rights which apply to all TF Git repositories.

Granting additional TF users Gerrit super user permissions

Log into Gerrit using the dedicated TeamForge account used by the Git integration. Click on Groups and select the Admistrators group.

After clicking on this Group you will able to add additional users which will have Gerrit super user permissions afterwards. Those users do not have to be site admins in TeamForge.

Adding internal groups and assigning users in Gerrit

You can make use of Gerrit's internal group feature if you like to assign additional permissions to a group of users. Refer to the follow up steps to enable CodeSearch integration for an example how this feature can be used to grant a certain groups of users read permissions to all Git repositories. By default, a group is owned by itself, so everybody in this group may add or delete additional users. If this is not desired, consider changing the owner of this group to another Gerrit group which contains only trusted users (e.g. Administrators group).

Adding additional (review related) access rights to mapped TeamForge project roles and TeamForge user groups

If you have Owner permissions for a Gerrit group associated with a Gerrit project (which will be the case if you belong to a project role with SCM admin permissions in the corresponding TeamForge Git repository), you can assign additional Gerrit access rights to arbitrary Gerrit groups (mapped TeamForge project roles, mapped TeamForge user groups, internal Gerrit groups) in your Gerrit project. However, changes to access rights of Gerrit groups managed by TeamForge (ie. Groups corresponding to TF project roles and user groups) will be overridden in the next permission synch **unless** the custom code review policy has been configured for the Git repository in question. If you want to make your changes survive, either use the custom review policy or an internal Gerrit group. Since Gerrit groups may contain other Gerrit groups, it is possible to add additional access rights on top of a TF project role by creating an internal gerrit group which includes the Gerrit group corresponding to the TF project role

History Protection

We define History Rewrite as non-fast forward updates of remote refs and its associated objects. This happens whenever a branch in the remote repository gets deleted, previously pushed commits get amended/tree filtered and forcefully re-pushed, or a remote branch/tag is pointed to an entire different commit history.
In other words, history may get rewritten without leaving any trace of the previous state. Sometimes, this behaviour is what the user wants, i.e. in case of removing code violating *intellectual property*, removing mistakenly committed large binary files or removing merged feature branches. This is why our TeamForge Git integration does not disable history rewrite completely but enables it for SCM Administrators only (see RBAC section of this document). However, rewriting history is a Git feature that can be easily abused and may result in accidental data loss. This is why we introduced our History Protection feature as a safety net and necessity for ensuring proper audit compliance.

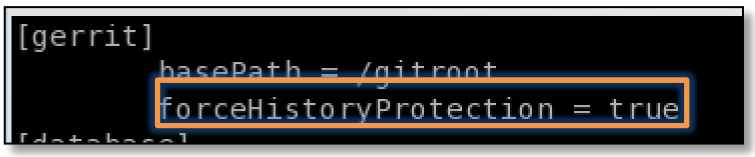
History Protection archives rewritten changes and keeps backups of deleted branches. If history changes occur, an immutable backup ref will be created in the remote repository, notification emails will be sent to all members of the Gerrit Administrators group and an event will be logged in audit log.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

Enable History Protection

There are two ways to turn on History Protection:

1. Integration server-wide (For all repositories hosted in this Git Integration Server)

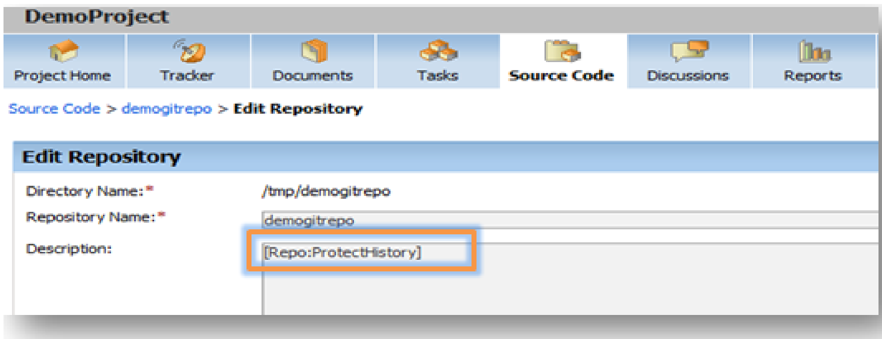
   Setting property `forceHistoryProtection=true` in `/opt/collabnet/gerrit/etc/gerrit.config` under `[gerrit]` section and restart gerrit running `$ service gerrit restart`

   ```
   [gerrit]
           basePath = /gitroot
           forceHistoryProtection = true
   [database]
   ```

2. Per Git Repository

   Go to TeamForge Project→Source Code→Select Git repository where History Protection is to be turned on


DemoProject

Project Home | Tracker | Documents | Tasks | Source Code | Discussions | Reports

Source Code > demogitrepo > Edit Repository

Edit Repository

Directory Name:* /tmp/demogitrepo

Repository Name:* demogitrepo

Description: [Repo:ProtectHistory]

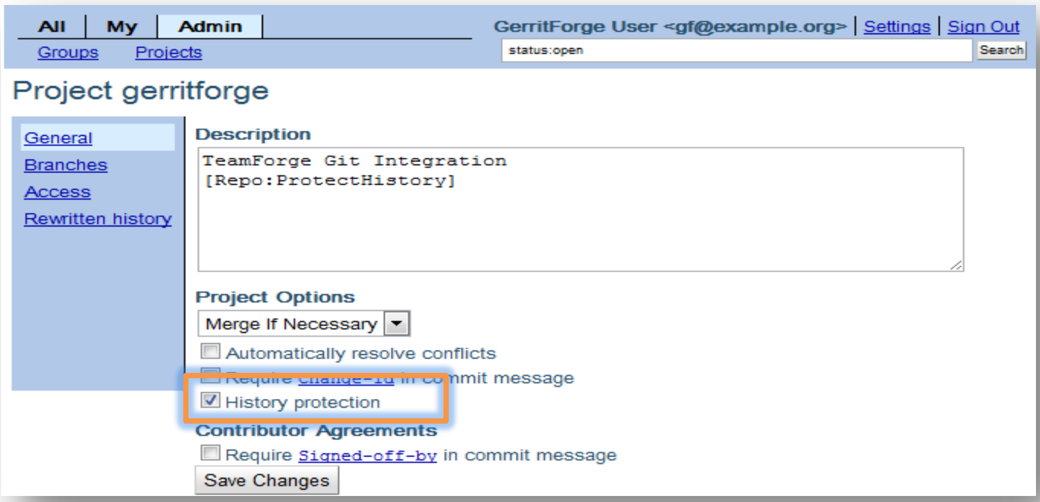
Make sure that your description field contains exact string "**[Repo:ProtectHistory]**" , and then press 'save'

NOTE: History Protection can be turned on / off any point of time. However, it will not be reflected in Gerrit immediately. It will be activated after time defined as regular refresh interval during Git Integration installation.

If you want to get this change in effect immediately, then right after changing repository description, user with SCM admin permission can do following:

'Remove any user from project role with any scm permission temporarily and add that user back'.

This action will trigger immediate sync, which will then enable 'History Protection' feature. After that, 'Gerrit Administrator' will be able to see history protection enabled in Gerrit WebUI (Login as 'Gerrit Administrator'→Project(Git Repository name in TeamForge)→General) as below.


All | My | Admin   GerritForge User <gf@example.org> | Settings | Sign Out

Groups Projects   status:open Search

Project gerritforge

General
Branches
Access
Rewritten history

Description

```
TeamForge Git Integration
[Repo:ProtectHistory]
```

Project Options

Merge If Necessary

☐ Automatically resolve conflicts

☐ Require Change-Id in commit message

☑ History protection

Contributor Agreements

☐ Require Signed-off-by in commit message

Save Changes

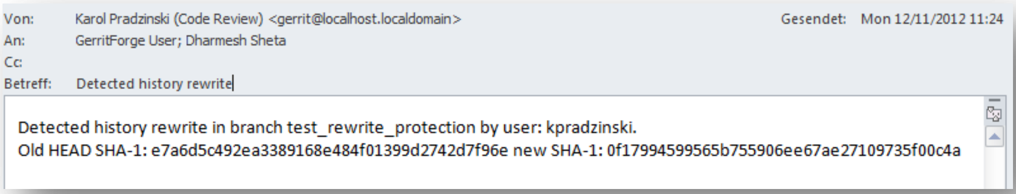
NOTE: If history protection is enabled Integration Server–wide, it cannot be overridden (turned off) per Git repository and will remain on for all Git repositories on given integration server.

History Protection Report

Once history protection is turned on, any 'non-fast forward' push to the remote repository or any remote branch/tag deletion will be recorded and reported.

Email Notification

In such event of 'history rewrite', an email will be sent out 'Administrator' group members in Gerrit.

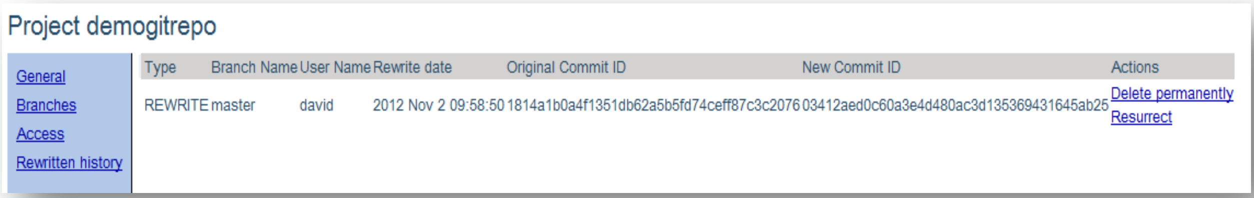


Von:	Karol Pradzinski (Code Review) <gerrit@localhost.localdomain>
Gesendet:	Mon 12/11/2012 11:24
An:	GerritForge User; Dharmesh Sheta
Cc:	
Betreff:	Detected history rewrite

```
Detected history rewrite in branch test_rewrite_protection by user: kpradzinski.
Old HEAD SHA-1: e7a6d5c492ea3389168e484f01399d2742d7f96e new SHA-1: 0f17994599565b755906ee67ae27109735f00c4a
```

Gerrit WebUI

Every 'history rewrite' event will be logged and stored in Gerrit database and visible in Gerrit WebUI*(Login as 'Gerrit Administrator'→Project→Rewritten History)*

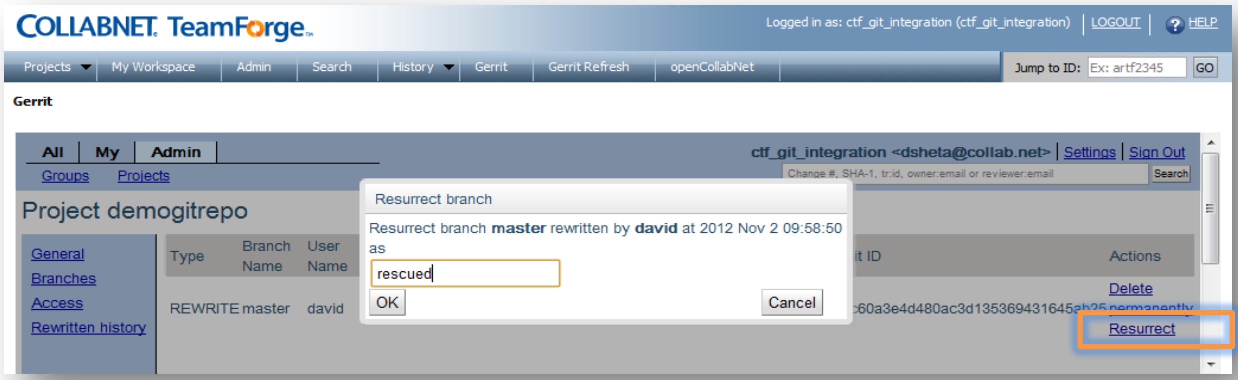


Project demogitrepo

General
Branches
Access
Rewritten history

Type	Branch Name	User Name	Rewrite date	Original Commit ID	New Commit ID	Actions
REWRITE	master	david	2012 Nov 2 09:58:50	1814a1b0a4f1351db62a5b5fd74ceff87c3c2076	03412aed0c60a3e4d480ac3d135369431645ab25	Delete permanently Resurrect

Gerrit Administrator can restore history by clicking 'resurrect' link . On this action, administrator will be asked to provide new name for the branch which will be created.
Backup branches can be permanently deleted by clicking 'Delete Permanently' (only if you are Gerrit Administrator).


COLLABNET TeamForge
Logged in as: ctf_git_integration (ctf_git_integration) | LOGOUT | HELP
Projects
My Workspace
Admin
Search
History
Gerrit
Gerrit Refresh
openCollabNet
Jump to ID: Ex: artf2345 GO
Gerrit
All
My
Admin
Groups
Projects
ctf_git_integration <dsheta@collab.net> | Settings | Sign Out
Change #, SHA-1, tr:id, owner:email or reviewer:email
Search

Project demogitrepo

Resurrect branch

Resurrect branch **master** rewritten by **david** at 2012 Nov 2 09:58:50 as

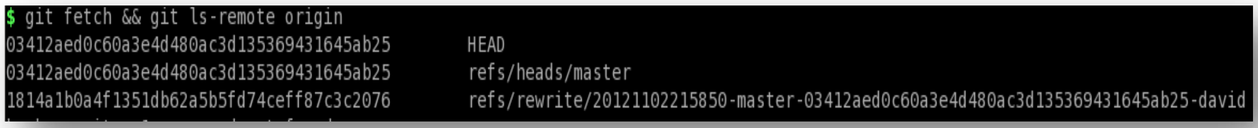
rescued

OK Cancel

Type	Branch Name	User Name	it ID	Actions
REWRITE	master	david	:60a3e4d480ac3d135369431645ab25	Delete permanently Resurrect

Git Command line

Users can also use their standard Git client(s) to find about about rewritten/deleted branches by running `git fetch && git ls-remote`



```
$ git fetch && git ls-remote origin
03412aed0c60a3e4d480ac3d135369431645ab25        HEAD
03412aed0c60a3e4d480ac3d135369431645ab25        refs/heads/master
1814a1b0a4f1351db62a5b5fd74ceff87c3c2076        refs/rewrite/20121102215850-master-03412aed0c60a3e4d480ac3d135369431645ab25-david
```

Entries in `refs/rewrite` (for non- fast forward) and `refs/delete` can be viewed using git command line `ls-remote` command only if read access to `refs/*` is granted. Gerrit will prevent any other action (like delete/force-update) on those special refs for all users (including administrators).

Audit log entries

Following events related to 'history protection' will be logged into audit log:

- Remote branches deleted
- History Re-written *(non fast forward push)*
- Backup branches resurrected
- Backup branches permanently deleted
- History Protection turned on/off

Which is located under `/opt/collabnet/gerrit/logs/gerrit.audit.log`

Scalability, hardware requirements

Please refer to http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/dev-design.html#_scalability for estimated requirements and availability assumptions. See http://code.google.com/p/gerrit/wiki/Scaling for hardware considerations.

The documentation also contains a detailed description of performance related settings: http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/config-gerrit.html

File / Directory locations, backup notes

Gerrit expects a standardized directory structure underneath the ***GERRIT_SITE*** directory *`(/opt/collabnet/gerrit`*, the gerrit user's home directory):

`/etc/default/gerritcodereview`: File containing GERRIT_SITE variable, points to *`/opt/collabnet/gerrit`*

Subdirectories of GERRIT_SITE (/opt/collabnet/gerrit):
`.ssh`: contains 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-CTF synchronization service
- *`gerrit.sh`*: SYSV-style init script; launches and shuts down, linked to *`/etc/init.d/gerrit`*

`cache`: disk cache. Does not needed to be backed up, always can 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`*: Gerrit's main configuration file
- *`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 and we keep 10 copies of it. Changes can be done in Gerrit's *`log4j.properties`* file in *`/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. Location can be moved by changing the setting in gerrit.config or by symlinking the directory. *Needs to be backed up.*

Database

Gerrit stores runtime information about users, groups and code reviews/commits in a Postgres database named *reviewdb*. This database needs to be backed up. If the Git integration is installed on the same host as TeamForge's Postgres DB, it will use the same Postgres install as CTF; if it is on a separate host, it will use a local installation on this host.
During the installation, new role *gerrit* and new schema *reviewdb* will be created. Note that at any point Git is not accessing TF schemas within Postgres DB. For accessing *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
```

In free form install mode, we do not have full control over the Postgres installation and `hb_pga.conf` may contain the following conflicting line:

```
host        all      all                  127.0.0.1/32        ident
```

Please comment(placing #) it if it exist to avoid clash with line (inserted by installer)

```
host      reviewdb          gerrit          samehost          md5
```

if you should encounter any problems with Gerrit connecting to its data base.

Open ports, connectivity requirements

Gerrit opens three (TCP) ports: `9080`, `9081` and `29418`. Only `29418` should be exposed outside localhost.

`9080`: The Gerrit Web Interface. This will be proxied by apache conf (prefix /gerrit) and doesn't need to be externally accessible. Port should not be exposed to outside.

`9081`: *Gerrit-sync web service (REST)*. In co-hosted mode, TF talks to Gerrit REST API over localhost. In distributed mode, TF talks to Gerrit REST API over an Apache proxy rule (SSL enabled if Apache is SSL enabled on the integration server where Gerrit is running). Port should not be exposed to outside.

`29418`: *Gerrit SSH access*. Developers use the SSH protocol to push/pull source code to/from Gerrit. This port needs to be open to users. NOTE: Gerrit ships its own SSH implementation and offers *no* shell access over this port.

Connectivity in co-hosted / local mode

In cohosted mode, TF talks to Gerrit REST API over localhost. Gerrit talks to TF over its default SOAP URL.

Connectivity in distributed/remote mode

The Git integration needs bidirectional connectivity to the TF host. In distributed mode, TF talks to 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 TF over its default SOAP URL.

Setting up Gerrit's integration with CI

For a nice introduction what Gerrit review feature is all about and how it works with CI, have a look at http://alblue.bandlem.com/Tag/gerrit/, http://urlenco.de/vhqjl and the official Gerrit documentation on http://source.android.com/source/life-of-a-patch.html and http://gerrit-documentation.googlecode.com/svn/Documentation/2.1.7/user-upload.html .

Setting up TeamForge Git Integration with Eclipse and Visual Studio

CollabNet's plugins for Eclipse and Visual studio integrate smoothly with TeamForge's Git integration. If you browse TeamForge's Git repositories, you will be able to clone them directly from your IDE. Eclipse will also detect Gerrit and exposes 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

gerritforge.mappings Backward compatibility notes

Existing customers are still able to use their old gerritforge.mappings without any behavioral changes if they do not want to benefit from the newly introduced code review policy feature. To determine whether a customer is still using the old gerritforge.mapping file format, we introduced one property called mapping_format_version. Its default value (if not mentioned) is 1. In the new template file it is set to 2. This will allow us change the format in the future again. If the number cannot be parsed, we should issue an error message and then assume 2.

The following two paragraphs are very technical and should not concern customers wanting to upgrade unless they have done extensive manual changes to access rights directly in Gerrit Web UI or use the programmatic access right infrastructure.

In backward compatibility mode (mapping_format_version < 2), the only managed categories are `READ, pHD, pTag, and OWN`.

Access rights of other categories (like submit, review, forge identity) will never be touched by GerritSynch. In `mapping_format_version == 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 will wipe 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 will be 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 backward compatibility mode is on, it will 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 is set to 2, we will keep the existing access rights no matter what.

Legacy Documentation

Git Integration v 6.2

TF objects and relationships mapped to Gerrit for TF Git Integration v6.2 only

TF Relationship	**Gerrit Relationship (also see table with category codes)**
SCM Repository in TF project with project roles with SCM permissions	*SCM Repository in TF project with project roles with SCM permissions*
User is part of a User Group which is associated with a Project Role	*User is part of Group(which corresponds to TF Project Role)*
User is a part of a User Group	*User is part of Group(which corresponds to TF User Group)*

Project Role is assigned an SCM Admin Permission	*Corresponding group is assigned Gerrit access rights (category code, lower range, upper range):*
	
	`READ,1,2`
	`pHD,1,3`
	`pHD,1,3,refs/tags/*`
	`pTAG,1,2`
	`OWN,1,1`

Project Role is assigned an SCM delete Permission	*Corresponding group is assigned Gerrit access rights (category code, lower range, upper range):*
	
	`READ,1,2`
	`pHD,1,3`
	`pTAG,1,2`

Project Role is assigned an SCM view & commit Permission	*Corresponding group is assigned Gerrit access rights(category code, lower range, upper range):*
	
	`READ,1,2`
	`pHD,1,2`

Project Role is assigned an SCM view Permission	*Corresponding group is assigned Gerrit access rights (category code, lower range, upper range):* `READ,1,1`
Project Role is assigned an SCM no Permission	*Corresponding group is assigned Gerrit access rights (category code, lower range, upper range):* `READ,-1,-1`

Registered user in TF who has logged into Gerrit at least once	*Implicit access rights in all Gerrit projects (category code, lower range, upper range):*
	`CRVW, -1, 1`
	`FORG, 1, 1`