## Wikiprint Book

**Title: Repository Administration**

**Subject: TechTIDE-Wiki - TracRepositoryAdmin**

**Version: 1**

**Date: 11/25/25 01:47:59**

## Table of Contents

# Repository Administration

## Quick start

- Enable the repository connector(s) for the version control system(s) that you will use.
- Add repositories through the *Repositories* admin panel, with `trac-admin` or in the `[repositories]` section of trac.ini.
- Set up a call to `trac-admin $ENV changeset added $REPO $REV` in the post-commit hook of each repository. Additionally, add a call to `trac-admin $ENV changeset modified $REPO $REV` in the post-revprop-change hook of repositories allowing revision property changes.
- Make sure the user under which your hooks are run has write access to the Trac environment, or use a tool like `sudo` to temporarily elevate privileges.

## Enabling the components

Support for version control systems is provided by optional components distributed with Trac, which are disabled by default *(since 1.0)*. Subversion and Git must be explicitly enabled if you wish to use them.

The version control systems can be enabled by adding the following to the `[components]` section of your trac.ini, or enabling the components in the *Plugins* admin panel.

```
tracopt.versioncontrol.svn.* = enabled

tracopt.versioncontrol.git.* = enabled
```

## Specifying repositories

Trac supports multiple repositories per environment, and the repositories may be for different version control system types. Each repository must be defined in a repository configuration provider, the two supported by default are the database store and the trac.ini configuration file. A repository should not be defined in multiple configuration providers.

It is possible to define aliases of repositories, that act as "pointers" to real repositories. This can be useful when renaming a repository, to avoid breaking links to the old name.

A number of attributes can be associated with each repository. The attributes define the repository's location, type, name and how it is displayed in the source browser. The following attributes are supported:

| Attribute | Description |
|---|---|
| `alias` | A repository having an `alias` attribute is an alias to a real repository. All TracLinks referencing the alias resolve to the aliased repository. Note that multiple indirection is not supported, so an alias must always point to a real repository. The `alias` and `dir` attributes are mutually exclusive. |
| `description` | The text specified in the `description` attribute is displayed below the top-level entry for the repository in the source browser. It supports WikiFormatting. |
| `dir` | The `dir` attribute specifies the location of the repository in the filesystem. It corresponds to the value previously specified in the option `[trac] repository_dir`. The `alias` and `dir` attributes are mutually exclusive. |
| `hidden` | When set to `true`, the repository is hidden from the repository index page in the source browser. Browsing the repository is still possible, and links referencing the repository remain valid. |
| `sync_per_request` | When set to `true` the repository will be synced on every request. This is not recommended, instead a post-commit hook should be configured to provide explicit synchronization and `sync_per_request` should be set to `false`. |
| `type` | The `type` attribute sets the type of version control system used by the repository. Trac supports Subversion and Git out-of-the-box, and plugins add support for many other systems. If `type` is not specified, it defaults to the value of the `[trac] repository_type` option. |
| `url` | The `url` attribute specifies the root URL to be used for checking out from the repository. When specified, a "Repository URL" link is added to the context navigation links in the source browser, that can be copied into the tool used for creating the working copy. |

A repository `name` and one of `alias` or `dir` attributes are mandatory. All others are optional.

For some version control systems, it is possible to specify not only the path to the repository in the `dir` attribute, but also a *scope* within the repository. Trac will then only show information related to the files and changesets below that scope. The Subversion backend for Trac supports this. For other types, check the corresponding plugin's documentation.

After adding a repository, the cache for that repository must be re-synchronized once with the `trac-admin $ENV repository resync` command.

`repository resync <repos>`

Re-synchronize Trac with a repository.

#### In `trac.ini`

Repositories and repository attributes can be specified in the `[repositories]` section of trac.ini. Every attribute consists of a key structured as `{name}.{attribute}` and the corresponding value separated with an equal sign (=). The name of the default repository is empty.

The main advantage of specifying repositories in `trac.ini` is that they can be inherited from a global configuration (see the global configuration section of TracIni). One drawback is that, due to limitations in the `ConfigParser` class used to parse `trac.ini`, the repository name is always all-lowercase.

The following example defines two Subversion repositories named `project` and `lib`, and an alias to `project` as the default repository. This is a typical use case where a Trac environment previously had a single repository (the `project` repository), and was converted to multiple repositories. The alias ensures that links predating the change continue to resolve to the `project` repository.

```
[repositories]
project.dir = /var/repos/project
project.description = This is the ''main'' project repository.
project.type = svn
project.url = http://example.com/svn/project
project.hidden = true

lib.dir = /var/repos/lib
lib.description = This is the secondary library code.
lib.type = svn
lib.url = http://example.com/svn/lib

.alias = project
```

Note that `name.alias = target` makes `name` an alias for the `target` repo, not the other way around.

#### In the database

Repositories can also be specified in the database, using either the "Repositories" admin panel under "Version Control", or the `trac-admin $ENV repository` commands.

The admin panel shows the list of all repositories defined in the Trac environment. It allows adding repositories and aliases, editing repository attributes and removing repositories. Note that repositories defined in `trac.ini` are displayed but cannot be edited.

The following trac-admin commands can be used to perform repository operations from the command line.

`repository add <repos> <dir> [type]`

Add a repository `<repos>` located at `<dir>`, and optionally specify its type.

`repository alias <name> <target>`

Create an alias `<name>` for the repository `<target>`.

`repository remove <repos>`

Remove the repository `<repos>`.

`repository set <repos> <key> <value>`

Set the attribute `<key>` to `<value>` for the repository `<repos>`.

Note that the default repository has an empty name, so it will likely need to be quoted when running `trac-admin` from a shell. Alternatively, the name "`(default)`" can be used instead, for example when running `trac-admin` in interactive mode.

### Repository caching

The Subversion and Git repository connectors support caching, which improves the performance browsing the repository, viewing logs and viewing changesets. Cached repositories must be synchronized; either explicit or implicit synchronization can be used. When searching changesets, only cached repositories are searched.

Subversion repositories are cached unless the type is `direct-svnfs`. Git repositories are cached when `[git]` cached_repository is `true`.

## Repository synchronization

Prior to 0.12, Trac synchronized its cache with the repository on every HTTP request. This approach is not very efficient and not practical anymore with multiple repositories. For this reason, explicit synchronization through post-commit hooks was added.

There is also new functionality in the form of a repository listener extension point *(IRepositoryChangeListener)* that is triggered by the post-commit hook when a changeset is added or modified, and can be used by plugins to perform actions on commit.

### Mercurial Repositories

Please note that at the time of writing, no initial resynchronization or any hooks are necessary for Mercurial repositories - see ■#9485 for more information.

### Explicit synchronization

This is the preferred method of repository synchronization. It requires setting the `sync_per_request` attribute to `false`, and adding a call to `trac-admin` in the `post-commit` hook of each repository. Additionally, if a repository allows changing revision metadata, a call to `trac-admin` must be added to the `post-revprop-change` hook as well.

`changeset added <repos> <rev> […]`

Notify Trac that one or more changesets have been added to a repository.

`changeset modified <repos> <rev> […]`

Notify Trac that metadata on one or more changesets in a repository has been modified.

The `<repos>` argument can be either a repository name (use "`(default)`" for the default repository) or the path to the repository.

Note that you may have to set the environment variable `PYTHON_EGG_CACHE` to the same value as was used for the web server configuration before calling `trac-admin`, if you changed it from its default location. See Trac Plugins for more information.

#### Subversion

#### Using `trac-svn-hook`

In a Unix environment, the simplest way to configure explicit synchronization is by using the ■contrib/trac-svn-hook script. `trac-svn-hook` starts `trac-admin` asynchronously to avoid slowing the commit and log editing operations. The script comes with a number of safety checks and usage advice. Output is written to a log file with prefix `svn-hooks-` in the environment `log` directory, which can make configuration issues easier to debug.

There's no equivalent `trac-svn-hook.bat` for Windows yet, but the script can be run by Cygwin's bash.

Follow the help in the documentation header of the script to configure `trac-svn-hook`. Configuring the hook environment variables is made easier in Subversion 1.8 by using the ■hook script environment configuration. Rather than directly editing `trac-svn-hook` to set the environment variables, they can be configured through the repository `conf/hooks-env` file. Replace the ■configuration section with:

```
export PATH=$PYTHON_BIN:$PATH
export LD_LIBRARY_PATH=$PYTHON_LIB:$LD_LIBRARY_PATH
```

and set the variables `TRAC_ENV`, `PYTHON_BIN` and `PYTHON_LIB` in the `hooks-env` file. Here is an example, using a Python virtual environment at `/usr/local/venv`:

```
[default]
TRAC_ENV=/var/trac/project-1
PYTHON_BIN=/usr/local/venv/bin
PYTHON_LIB=/usr/local/venv/lib
```

#### Writing Your Own Hook Script

The following examples are complete post-commit and post-revprop-change scripts for Subversion. They should be edited for the specific environment, marked executable (where applicable) and placed in the `hooks` directory of each repository. On Unix (`post-commit`):

```
#!/bin/sh
export PYTHON_EGG_CACHE="/path/to/dir"
/usr/bin/trac-admin /path/to/env changeset added "$1" "$2"
```

Note: Check with `whereis trac-admin`, whether `trac-admin` is really installed under `/usr/bin/` or maybe under `/usr/local/bin/` and adapt the path. On Windows (`post-commit.cmd`):

```
@C:\Python26\Scripts\trac-admin.exe C:\path\to\env changeset added "%1" "%2"
```

The post-revprop-change hook for Subversion is very similar. On Unix (`post-revprop-change`):

```
#!/bin/sh
export PYTHON_EGG_CACHE="/path/to/dir"
/usr/bin/trac-admin /path/to/env changeset modified "$1" "$2"
```

On Windows (`post-revprop-change.cmd`):

```
@C:\Python26\Scripts\trac-admin.exe C:\path\to\env changeset modified "%1" "%2"
```

The Unix variants above assume that the user running the Subversion commit has write access to the Trac environment, which is the case in the standard configuration where both the repository and Trac are served by the web server. If you access the repository through another means, for example `svn+ssh://`, you may have to run `trac-admin` with different privileges, for example by using `sudo`.

See the ■section about hooks in the Subversion book for more information. Other repository types will require different hook setups.

**Git**

Git hooks can be used in the same way for explicit syncing of Git repositories. If your git repository is one that gets committed to directly on the machine that hosts trac, add the following to the `hooks/post-commit` file in your git repo (note: this will do nothing if you only update the repo by pushing to it):

```
#!/bin/sh
REV=$(git rev-parse HEAD)
trac-admin /path/to/env changeset added <repos> $REV
```

Alternately, if your repository is one that only gets pushed to, add the following to the `hooks/post-receive` file in the repo:

```
#!/bin/sh
tracenv=/path/to/env     # change with your Trac environment's path
repos=                   # change with your repository's name
while read oldrev newrev refname; do
   if [ "$oldrev" = 0000000000000000000000000000000000000000 ]; then
       git rev-list --reverse "$newrev" --
   else
       git rev-list --reverse "$newrev" "^$oldrev" --
   fi | xargs trac-admin "$tracenv" changeset added "$repos"
done
```

The `<repos>` argument can be either a repository name (use "`(default)`" for the default repository) or the path to the repository.

**Mercurial**

For Mercurial, add the following entries to the `.hgrc` file of each repository accessed by Trac (if ■TracMercurial is installed in a Trac `plugins` directory, download ■hooks.py and place it somewhere accessible):

```
[hooks]
; If mercurial-plugin is installed globally
commit = python:tracext.hg.hooks.add_changesets
changegroup = python:tracext.hg.hooks.add_changesets

; If mercurial-plugin is installed in a Trac plugins directory
commit = python:/path/to/hooks.py:add_changesets
```

```
changegroup = python:/path/to/hooks.py:add_changesets

[trac]
env = /path/to/env
trac-admin = /path/to/trac-admin
```

#### Per-request synchronization

If the post-commit hooks are not available, the environment can be set up for per-request synchronization. In that case, the `sync_per_request` attribute for each repository in the database and in trac.ini must be set to `false`.

Note that in this case, the changeset listener extension point is not called, and therefore plugins using it will not work correctly.

### Automatic changeset references in tickets

You can automatically add a reference to the changeset as a ticket comment whenever changes are committed to the repository. The description of the commit needs to contain one of the following formulas:

- **`Refs #123`** - to reference this changeset in #123 ticket
- **`Fixes #123`** - to reference this changeset and close #123 ticket with the default status *fixed*

This functionality requires installing a post-commit hook as described in #ExplicitSync, and enabling the optional commit updater components by adding the following line to the `[components]` section of your trac.ini, or enabling the components in the *Plugins* admin panel.

```
tracopt.ticket.commit_updater.* = enabled
```

For more information, see the documentation of the `CommitTicketUpdater` component in the *Plugins* admin panel and the ■CommitTicketUpdater page.

### Troubleshooting

#### My trac-post-commit-hook doesn't work anymore

You must now use the optional components from `tracopt.ticket.commit_updater.*`, which you can activate through the Plugins panel in the Administrative part of the web interface, or by directly modifying the [components] section in the trac.ini. Be sure to use explicit synchronization as explained above.

See ■CommitTicketUpdater#Troubleshooting for more troubleshooting tips.