Wikiprint Book

Title: Repository Administration

Subject: YAM - Yet Another Mailer - TracRepositoryAdmin

Version: 6

Date: 26.11.2014 04:05:19

Table of Contents

Repository Administration **3**
Quick start 3
Specifying repositories 3
In trac.ini 3
In the database 4
Repository synchronization 4
Mercurial Repositories 4
Explicit synchronization 4
Per-request synchronization 6
Migration from a single-repository setup (Subversion) 6
Migration from a single-repository setup (Mercurial) 6
Troubleshooting 6
My trac-post-commit-hook doesn't work anymore 6

Repository Administration

Quick start

- Manage repositories in the "Repository" 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.
- Set the `[trac] repository_sync_per_request` option to an empty value to disable per-request syncing.
- Make sure the user under which your Subversion hooks are run has write access to the Trac environment, or use a tool like `sudo` to temporarily elevate privileges.

Specifying repositories

Starting with 0.12, Trac can handle more than one repository per environment. The pre-0.12 way of specifying the repository with the `repository_dir` and `repository_type` options in the `[trac]` section of trac.ini is still supported, but two new mechanisms allow including additional repositories into an environment.

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

A number of attributes can be associated with each repository, which 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.
`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.

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 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 `[trac] repository_sync_per_request` option in trac.ini to an empty value, 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.

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: Ubuntu doesn't seem to like /usr/bin/trac-admin, so just use:

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

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`.

Note that calling `trac-admin` in your Subversion hooks can slow down the commit and log editing operations on the client side. You might want to use the ?contrib/trac-svn-hook script which starts `trac-admin` in an asynchronous way. The script also comes with a number of safety checks and usage advices which should make it easier to set up and test your hooks. There's no equivalent `trac-svn-hook.bat` for Windows yet, but the script can be run by Cygwin's bash.

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

Git hooks can be used in the same way for explicit syncing of git repositories. Add the following to `.git/hooks/post-commit`:

```
REV=$(git rev-parse HEAD)
trac-admin /path/to/env changeset added <my-repository> $REV
```

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 `[trac] repository_sync_per_request` option in trac.ini must be set to a comma-separated list of repository names to be synchronized.

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

Migration from a single-repository setup (Subversion)

The following procedure illustrates a typical migration from a Subversion single-repository setup to multiple repositories.

i. Remove the default repository specification from the `[trac] repository_dir` option.
ii. Add the main repository as a named repository.
iii. Re-synchronize the main repository.
iv. Set up post-commit and post-revprop-change hooks on the "main" repository, and set `[trac] repository_sync_per_request` to an empty value.
v. Add an alias to the main repository as the default repository (by leaving out the the `name`, e.g. `.alias = main`). This ensures that all links predating the migration still resolve to the main repository.
vi. Repeat steps 2, 3 and 4 to add other "named" repositories as needed.

Migration from a single-repository setup (Mercurial)

The following procedure illustrates a typical migration from a Mercurial single-repository setup to multiple 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.

i. Upgrade to the latest version of the TracMercurial plugin.
ii. Remove the default repository specification from the `[trac] repository_dir` option.
iii. Add the main repository as a named repository.
iv. Add an alias to the main repository as the default repository (by leaving out the the `name`, e.g. `.alias = main`). This ensures that all links predating the migration still resolve to the main repository.
v. Repeat step 3 to add other "named" repositories as needed.

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.