TortoiseHg Documentation

Release 2.0.0

Steve Borho and others

March 11, 2011

CONTENTS

CHAPTER

ONE

PREFACE

1.1 Audience

This book is written for computer literate folk who want to use Mercurial to manage their data, but are uncomfortable using the command line client to do so. Since TortoiseHg is a Windows shell extension it's assumed that the user is familiar with the Windows explorer and knows how to use it.

You can find the most up to date version of this documentation at our web site.

1.2 Reading guide

This Preface explains a little about the TortoiseHg project, the community of people who work on it, and the licensing conditions for using it and distributing it.

The *Introduction* explains what TortoiseHg is, what it does, where it comes from and the basics for installing it on your PC.

A Quick Start Guide to TortoiseHg is a quick tutorial on how to start with TortoiseHg.

TortoiseHg in daily use is the main chapter, it describes the frequently used components of TortoiseHg.

Settings describes how to configure TortoiseHg.

Use with other VCS systems describes how to use TortoiseHg as a client for non-Mercurial servers.

Frequently Asked Questions has a list of common questions and their answers.

Debugging describes how to debug any problems that you find.

1.3 TortoiseHg is free!

TortoiseHg is released under GPLv2. You are free to install it on as many computers as you like, and to redistribute it according to the GPLv2 license.

1.4 Community

Mailing Lists:

- Users - Announcements, user Q&A, and feature discussions.

- Developers - Patches, bug reports, development discussions.
- Issues - Notifications from the issue tracker.

And our wiki on Bitbucket.

1.5 Acknowledgements

Thanks to the many people who contribute to the TortoiseHg project. It takes a community of developers, translators, and users to build a truly useful tool (especially users who care enough to report bugs and file feature requests).

The TortoiseHg installer for Windows includes the TortoiseOverlays handler, as provided by the TortoiseSVN project.

The history viewer of TortoiseHg is based on hgview, a tool developed by David Douard and others, with the financial support of Logilab.

1.6 Conventions used in this manual

The following typographical conventions are used in this manual:

`Ctrl-A` Indicates a key, or combination of keys, to press.

Commit Indicates a label, button or anything that appears in user interfaces.

TortoiseHg... → ***About*** Indicates a menu choice, or a combination of menu choice, tab selection and GUI label. For example *TortoiseHg...* → *Global settings* → *Commit* → *User name* means do something in **User name** label under **Commit** tab selectable from the menu choice *TortoiseHg...* → *Global settings*.

`.hg/hgrc` Indicates a filename or directory name.

thg log Indicates a command to enter into command window.

`myproxy:8000` Indicates a text to enter into a text input field in the GUI.

Note: This is a note.

> **Warning:** An important note. Pay attention.

CHAPTER

TWO

INTRODUCTION

2.1 What is TortoiseHg?

TortoiseHg is a set of graphical tools and a shell extension for the Mercurial distributed revision control system.

Note: hg is the symbol for the chemical element Mercury

On Windows, TortoiseHg consists of the Workbench graphical application, a shell extension which provides overlay icons and context menus in your file explorer, and a command line program named `thg.exe` which can launch the TortoiseHg tools. Binary packages of TortoiseHg for Windows come with Mercurial and a merge tool and are thus completely ready for use "Out of the Box".

On Linux, TortoiseHg consists of a command line thg script and a Nautilus extension which provides overlays and context menus in your file explorer. You must have Mercurial installed separately in order to run TortoiseHg on Linux. TortoiseHg binary packages list Mercurial as a dependency, so it is usually installed for you automatically.

Note: On Windows, TortoiseHg includes both thg.exe and thgw.exe. The latter is intended to be launched from desktop shortcuts or menu entries as it will refuse to open a command console. thg.exe is for use on the console, and can provide command line help. The thg.cmd batch file that our installer adds into your PATH runs thg.exe.

TortoiseHg is primarily written in Python and PyQt (the Windows shell extension being the notable exception). The thg script and TortoiseHg dialogs can be used on any platform that supports PyQt, including Mac OS X.

2.2 Installing TortoiseHg

2.2.1 On Windows

TortoiseHg comes with an easy to use MSI installer. You can always find the most up to date release on our website. Simply double click on the installer file and follow its instructions.

After a first time install, a re-login is usually required to start the icon overlays.

During upgrades, the installer will ask to close or restart any applications that have loaded the TortoiseHg shell extension. If you allow those applications to be closed, the upgrade will not require a reboot or logout. If other users are logged in, or if there are applications which cannot be shutdown, a reboot will be required to complete the install.

Note: If you have a legacy version of TortoiseHg installed, the installer will require that you to remove it. The uninstall can be initiated from the control panel or the start menu.

Warning: Legacy uninstallers (<=0.9.3) have a tendency to delete your user Mercurial.ini file, so backup your file before uninstalling the older TortoiseHg versions. This is not a problem with the newer MSI packages.

Legacy TortoiseHg installers (prior to version 1.0) were built with InnoSetup. They installed a TortoiseOverlay package as a separate application, so you always saw both TortoiseHg and TortoiseOverlay as two applications in the *Add/Remove Programs* control panel program. (On x64 platforms, there were two TortoiseOverlays, one for x86 processes and one of x64 processes).

The new MSI installers for TortoiseHg include the TortoiseOverlay packages as "merge modules" so they do not appear as separate applications anymore. It should be safe to uninstall the older TortoiseOverlay applications from *Add/Remove Programs* after you uninstalled the legacy (<=0.9.3) TortoiseHg installer, unless you have other Tortoise products that still use the separate TortoiseOverlay MSI approach (TortoiseCVS or TortoiseBZR).

Note: TortoiseOverlay is a shim package that allows multiple Tortoise style shell extension clients to share overlay slots. This is necessary because even modern Windows platforms only support a limited number of overlay slots (11-14). TortoiseOverlay packages are created by the TortoiseSVN developers.

To be completely safe, there are two approaches you can take:

1. Just leave the old TortoiseOverlay packages installed. They do not harm anything.
2. Uninstall all the old TortoiseOverlay packages, then re-install all of your Tortoise products until they are all functional.

Finally, if you have problems with the shell extension even after re-logging in, we recommend you re-run the installer and select the *Repair* option. There were problems with a few versions of TortoiseOverlay that cause upgrades to subtly fail until the *Repair* process is run.

Language settings

The TortoiseHg user interface has been translated into many languages. Language packs are not required since all available languages are installed by default. You can select your preferred **UI Language** in the global settings tool.

The Windows shell extension context menus get their translations from the Windows registry. Translations for many locales were installed under `C:\Program Files\TortoiseHg\i18n\cmenu`. Select the locale you would like to use, double-click on it, and confirm all requests.

2.2.2 On Linux and Mac

The most recent Linux packages can be found on our download page.

For Mac OS X, no packages are available but you can run thg and all the dialogs via the source install method. For details, see Mac OS X.

Note: If you install TortoiseHg from source, you need to add our `contrib/mergetools.rc` file to your HGRC path in some way. One approach is to *%include* it from your `~/.hgrc file`.

CHAPTER

THREE

WHAT'S NEW

3.1 TortoiseHg 2.0

3.1.1 Philosophy

The following philosophical changes were made between TortoiseHg 1.0 and TortoiseHg 2.0.

Workbench

We wanted a single 'TortoiseHg' application which can access nearly all TortoiseHg (and Mercurial) functionality and that could be launched by a desktop or start menu shortcut. So we developed the Workbench application.

The Workbench can support multiple repositories open at a time via "Repo Tabs" across the top of the main window.

Each repository tab supports multiple "Task Tabs" beneath its graph splitter. These task tabs are switchable via icons on the side of the Workbench or via application menus. Available task tabs include a changeset browser, a commit tool, a manifest browser, a history search widget, and a sync widget.

Also available are two dockable widgets - a *Repository Registry* which lists all known repositories on your local machine and an *Output Log Window* which displays running command lines and their output and can also function as a minimal shell.

Showing Mercurial command lines

In an effort to educate users on Mercurial's command interface, nearly all commands are executed in the log window, displaying the full command line and Mercurial's output (progress indication is provided by progress bars inside the Workbench status bar). The few tools that do not use a command log window will generally display the command line they execute.

Resolve tool, deliberate merges

TortoiseHg 2.0 introduces a resolve dialog for resolving conflicted file merges. It shows the users all the files that require resolution and those files that have been resolved, allowing merges to be verified.

As supported by Mercurial's resolve command, individual file merges may be restarted as many times as necessary to get the merge correctly completed.

By default, TortoiseHg will use the resolve dialog to resolve all conflicts, including trivial conflicts. It instructs Mercurial to never merge files automatically, deferring their resolution until the resolve dialog can be launched. This is true for merges, update commands that require content merges, rebases, and backouts.

Shelve Improved

TortoiseHg 2.0 includes a new shelve tool which is capable of moving changes between your working directory, a shelf file, or an unapplied MQ patch.

Revision Sets

We have replaced the filter bar of the Repository Explorer with a revision set bar in the Workbench. Revision sets were introduced in Mercurial 1.6 and have been integrated with an increasing number of commands in each subsequent release. They are a powerful query language for finding and specifying revisions in your repository.

The Workbench also includes a revision set editor which both teaches the user the available keywords and their arguments, and offers brace matching, auto-completions, and other editing amenities.

In TortoiseHg 2.0, incoming and outgoing changesets are visualized as revision sets. In previous versions they were represented by graph annotations.

3.1.2 Technology

Qt and PyQt

TortoiseHg 2.0 has been a near rewrite of all of the tools and dialogs taking advantage of Nokia's excellent Qt UI framework and Riverbank Computing's fine PyQt Python bindings.

QScintilla2

TortoiseHg uses the QScintilla2 editing component extensively to:

- display file contents and diffs with syntax highlighting
- display annotations with syntax highlighting
- edit commit messages with auto-completion of filenames and source symbols
- edit revision set strings with brace matching and auto-completion

One can configure the QScintilla2 tab stop parameter using the settings tool, while white space visibility and wrap are controlled by context menus.

Polling of repository state and config

The Workbench and other applications like the commit tool will poll repositories on your local machine to detect changes made to either the repository or their configuration files, and automatically update running applications as necessary. Nearly all configuration changes are effective immediately, with the notable exception of enabling or disabling Mercurial extensions. Changes to extension configuration generally require application restarts before they take effect.

Immediate bug report dialogs

Prior to TortoiseHg 2.0, bug reports were written to stderr as they occured and stderr was captured and scanned at exit to report those errors to the user. While we gained many valuable bug reports via this mechanism, there was rarely any context on what operations caused these bugs.

In TortoiseHg 2.0, we have created a generic exception handler that catches all Python exceptions that are otherwise unhandled by application code. This allows us to display exception tracebacks almost immediately after they occur (after a short timeout to collect consecutive exceptions together). The hope is that future bug reports will contain better reproduction instructions, or at least context for the tracebacks.

Demand loaded graph

To keep refreshes as efficient as possible, the graphing algorithm will only load a couple hundred revisions initially during a refresh, and then load further revisions only when those revisions are required to be displayed. You will notice scrolling through the graph is jerky, these are bulk loads of revisions into the graph. To avoid this jerkiness you can force TortoiseHg to load all revisions in the graph via the **Load all revisions** option from the **View** menu.

CHAPTER

FOUR

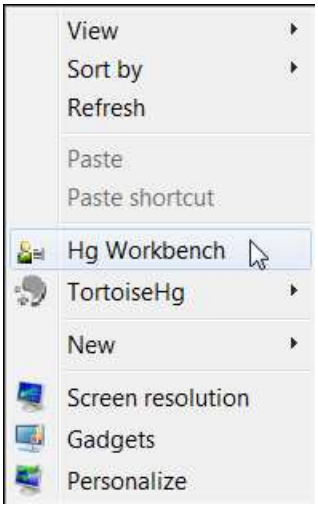
A QUICK START GUIDE TO TORTOISEHG

Welcome to TortoiseHg and the Mercurial! TortoiseHg is a set of graphical applications and Windows Explorer shell extension that serve as a friendly front-end to the Mercurial distributed version control system (DVCS).

All TortoiseHg functionality is reachable from 3 places:

1. The **Workbench** application

   You can start the Workbench from the Start Menu, or by right clicking on the Desktop and selecting *Hg Workbench*.



   Start the Workbench from the Desktop

2. The **Explorer** context menu

   All you have to do is right click on the right folder or files in Explorer, and select a context menu entry.

3. The **thg** command line application

   Type the appropriate commands from any command line interface, in the form `thg <command> [options]`.

In this quick guide we would like to make you get started using the Workbench application, but we will also indicate how to do the same with the other possibilities.

Mercurial commands are also available from the standard **hg** command line application.

4.1 Configuring TortoiseHg

Your first step should be to make sure that you are correctly identified to TortoiseHg. You do this by opening the global settings dialog.

Workbench: select *File* → *Settings...* from the menu

Explorer: choose *TortoiseHg* → *Global Settings* from the context menu

Command line: type **thg userconfig**

This opens the TortoiseHg settings dialog, editing your global (user) configuration.

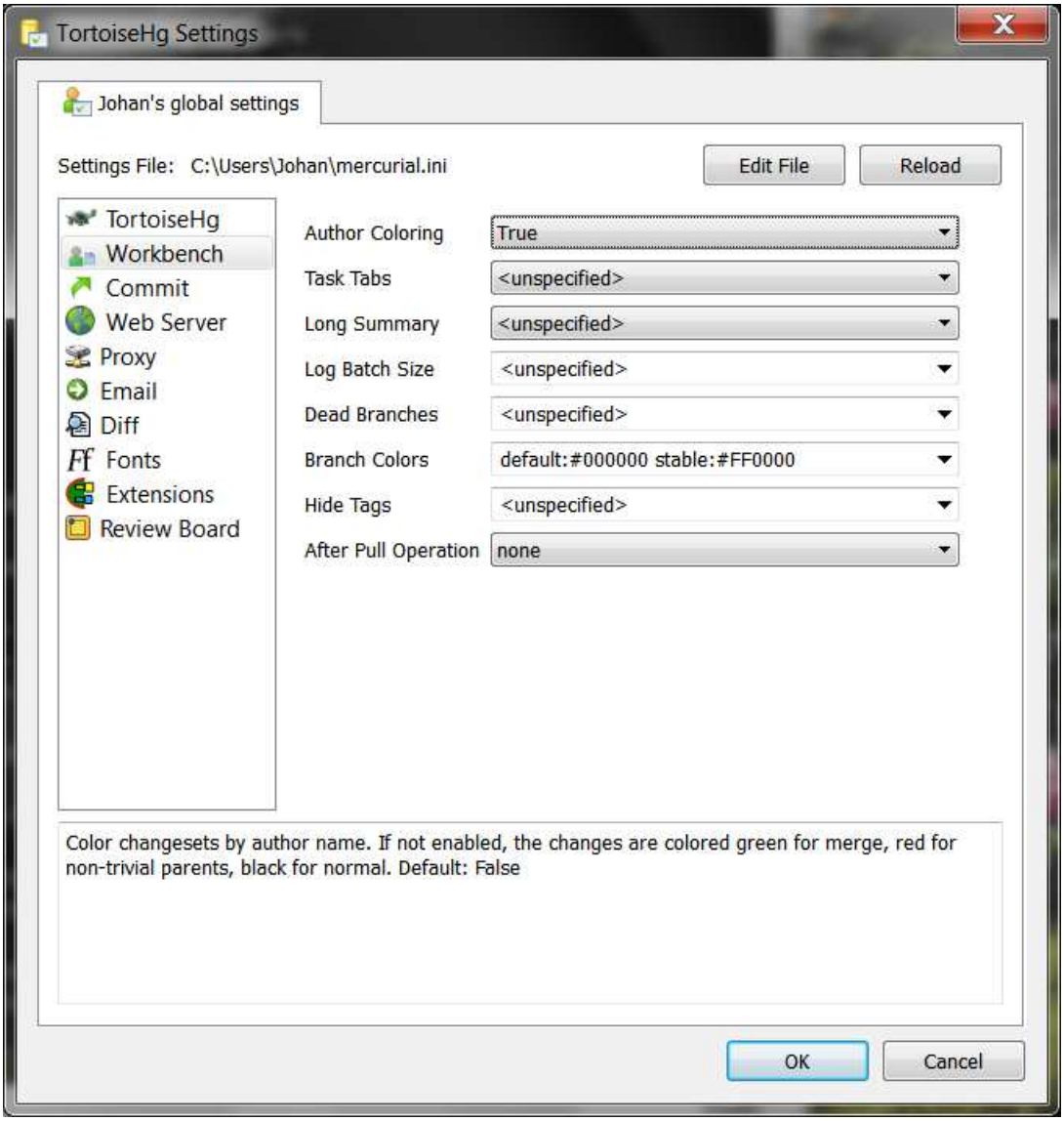


Figure 4.1: TortoiseHg Settings Dialog

First select the **Commit** page and enter a name in the **Username** field.

Note: If you neglect to configure a username TortoiseHg will ask you to enter one when you try to *commit*, which is the first time a username is actually required.

Note: There are no hard fast rules on how to format your username, the field is free form, but the following convention is commonly used:

```
FullName <email>
```

for example

```
Donald Duck <donaldduck@example.net>
```

The email address is stripped when viewing history in the revision history viewer, and the built-in web server obfuscates email addresses to prevent SPAM.

Next, select the **TortoiseHg** page and select the **Three-way Merge Tool** entry. In the drop down list you will find all of the merge tools detected on your computer (kdiff3 is provided by the Windows installer) and a number of internal merge behaviors. Select your preferred merge tool.

If you prefer for TortoiseHg to also use your selected merge tool for visual diffs, you can leave the **Visual Diff Tool** unspecified. Otherwise, select your favorite visual diff tool from the drop down list of detected visual diff tools.

If there are no options in either drop-down list, you must install a diff/merge tool that is supported by our mergetools.rc or configure your own tool.

Note: If you installed TortoiseHg from source, you need to add our `contrib/mergetools.ini` file to your HGRC path in some way. One approach is to *%include* it from your `~/.hgrc` file.

Feel free to configure other global settings while you have the dialog open. You will have the chance later to override these global settings with repository local settings, if required.

Click the **OK** button to save the changes you have made and close the settings dialog.

Note: Most TortoiseHg settings changes are noticed immediately, but loading or unloading extensions usually requires restarting all open applications for the changes to take effect.

4.2 Getting Acquainted

Mercurial supports many different collaboration models. This chapter describes just one of those models: a single central repository. The central repository model does not scale as well as other models, but it is the most familiar to those coming from other revision tools and thus is the most common approach people start with.

To get started, suppose you volunteer to create the central repository. There are ways to convert non-Mercurial repositories into Mercurial repositories, but this example assumes you are starting from scratch.

4.3 Initialize the repository

Create the initial repository on your local machine:

Workbench: select *File → New Repository...* from the menu

Explorer: select *TortoiseHg → Create Repository Here* from the context menu

Command line: type **thg init**

You can do this from within the folder you want to create the repository in, or enter the correct path in the dialog. You only need to do this once in the root folder of your project.

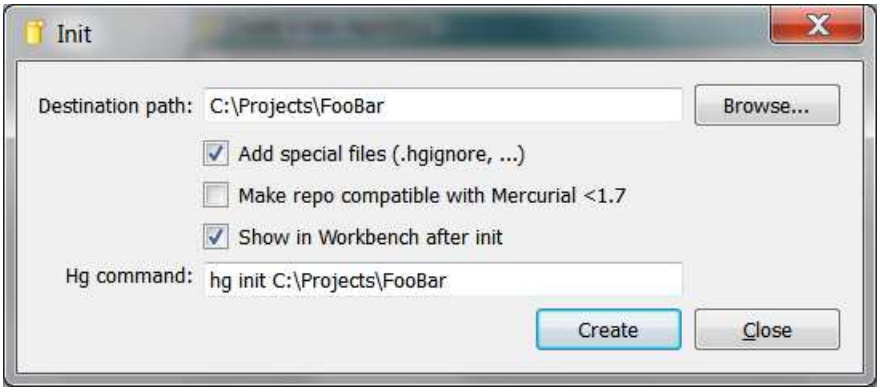


Figure 4.2: Repository Init Dialog

We suggest you keep **Add special files (.hgignore, ...)** checked, and do not check **Make repo compatible with Mercurial < 1.7**, unless you have a strong reason to do so.

After pressing **Create**, Mercurial creates a subdirectory in your project folder named `.hg`. This is where Mercurial keeps all its version data. It is called the *repository* or *store*, while the directory containing the source files is called the *working directory*. You never need to specify the `.hg` directory when running commands, you only need to specify the working directory root. It is mentioned here just so you better understand how Mercurial works.

The new repository will also be added to the RepoRegistry when you perform this operation from the Workbench.

Warning: It is dangerous to manually edit the files in `.hg` directory, repository corruption can occur. `.hg/hgrc` is perhaps the only exception to this rule.

Note: Perhaps you already created one or more repositories. As you can manage multiple repositories in the Workbench at the same time, you can add these existing repositories by selecting *File → Open Repository...* from its menu, and selecting their folder. Or you could drag their folder from Explorer into the RepoRegistry pane.

4.4 Add files

Now it's time to tell Mercurial which files must be tracked. There are various ways to do this:

1. Workbench: goto the Commit task tab, rightclick on the file, and select *Add* from the context menu. This will change the status indication of that file into 'A' and the filename will turn green.
2. Explorer: select *TortoiseHg → Add Files...* in the context menu. A dialog will open for you to check the selected files and accept the add operation. You can also open the status tool by selecting *TortoiseHg → View File Status*. Check the files you want to add and select **Add** from the file context menu.
3. Command line: type **thg status**
4. Or skip adding new files as a separate step and have the commit tool add them implicitly. The commit tool is very similar to the status tool and allows you to do all of the same tasks. In this tool you can add and commit an untracked file by just checking the file and pressing **Commit**.

4.5 Ignore files

You may have some files in the foldertree of your repository that you don't want to track. These can be intermediate results from builds f.i. that you do not wish to always delete immediately, or files your editor generates, etc. You can mark these files as ignored in some different ways too.

1. Workbench: goto the Commit task tab, rightclick on the file, and select *Ignore...* from the context menu to open the ignore filter dialog.
2. Explorer: select *TortoiseHg → Edit Ignore Filter*.
3. Command line: type **thg hgignore** to bring up the ignore filter dialog.
4. You can also launch the ignore filter from the status tool (the menu option is named **Ignore**).

Choose a file from the list or manually type in a *Glob* or *Regular expression* filter and then press **Add**. Changes to the ignore filter take effect immediately.

Note: The `.hgignore` file, contained in the working directory root, is typically tracked (checked in).

Note: It is good practice to not have many *unknown* files in your working directory, as it makes it too easy to forget to add vital new files. It is recommended that you keep your `.hgignore` file up to date.

4.6 Commit

Commit your local repository now:

Workbench: click on the Working Directory revision which also selects the Commit task tab, or directly select the Commit task tab

Explorer: right-clicking anywhere in the folder, or on the folder itself, and then selecting **Hg Commit...**

Command line: type **thg commit**

Write a commit message, select the files you wish to commit, then press **Commit**. Your previous commit message will be in the message history drop-down, so you do not have to type it in again from scratch.

4.7 Share the repository

Now you are ready to share your work. You do this by making a copy of your repository in a public location that everyone in your group can read. Mercurial calls this copy operation *cloning your repository*.

To clone your repository to a shared drive:

Workbench: select *File → Clone Repository...* from the menu

Explorer: select *TortoiseHg → Clone...* from the context menu

Command line: type **thg clone**

Then enter the destination path, and click **Clone**.

When you create a clone for the purposes of generating a *central repository* there is no reason for that clone to have a working directory. Checking **Do not update the new working directory** under **Options** will prevent Mercurial from checking out a working copy of the repository in the central repository clone. It will only have the `.hg` directory, which stores the entire revision history of the project.

Other team members can clone from this clone with or without a checked out working directory.

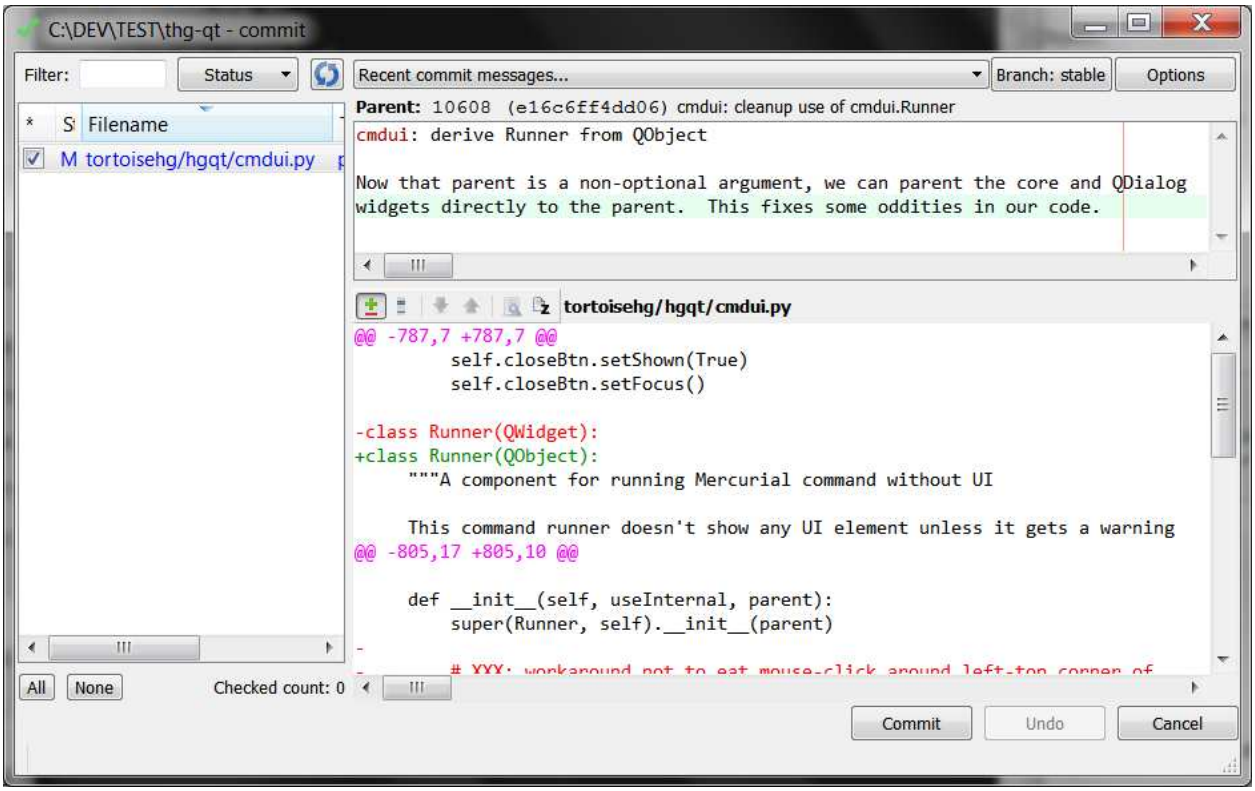


Figure 4.3: Commit Tool

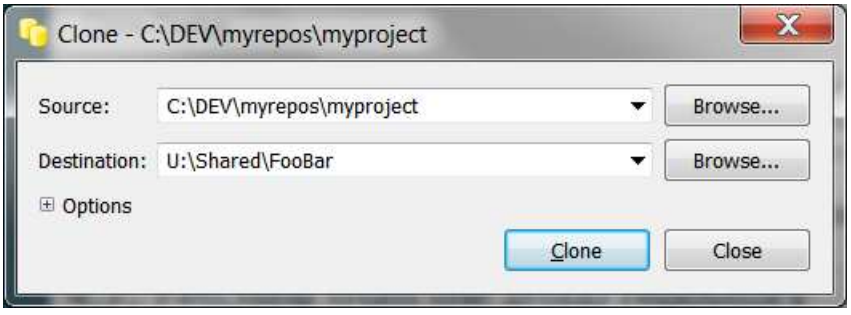


Figure 4.4: Clone Dialog

4.8 Fetching from the group repository

You want to start collaborating with your team. They tell you something like *fetch the repository from x*. What does that mean? It means that you want to make a copy of the repository located at x on your local machine. Mercurial calls this cloning and TortoiseHg has a dialog for it.

Workbench: select *File* → *Clone Repository...* from the menu

Explorer: select *TortoiseHg* → *Clone...* from the context menu

Command line: type **thg clone**

Then enter the destination path, and click **OK**.

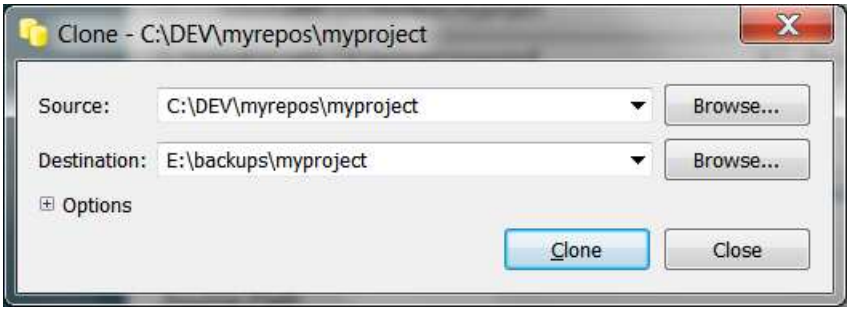


Figure 4.5: Clone Dialog

This time you do want to update the working directory because you want to work on the project, under **Options** uncheck **Do not update the new working directory** so Mercurial updates the working directory with the *tip* revision in your new clone.

4.9 Working with your repository

Suppose you've introduced some changes. It is easy to discover what pending changes there are in the repository.

Workbench: go to the Commit task tab and inspect the filelist at the left

Any files marked with 'A' (added, green), with '?' (unversioned but not ignored, fuchsia), with 'M' (modified, blue), or with '!' (removed, red) indicate pending changes that should be committed.

The Commit task tab in the Workbench gives you a way to see differences within the files, or you can use your visual difference tool (kdiff3). Mercurial allows you to commit many changes before you decide to synchronize (share changes) with the group repository.

Explorer: inspect the icons on the folders and files in your repository

Folders or files in Explorer marked with one of the icons below are another way of indicating pending changes. You can traverse the directories to find specific changes and commit them from Explorer. Though it might be quicker to do that from the Commit task tab in the Workbench.

Command line: type **thg commit**

When you're ready to publish your changes, you

1. Commit your changes to your local repository (see above).
2. Pull changes from the group repository into your repository using *TortoiseHg* → *Workbench* or **thg log**, select the Sync task tab, choose the path to the group repository in the syncbar and then click the **Pull** button.

Figure 4.6: Overlay Icons on Vista

3. If some changesets were pulled, merge those changes with your local changes and then commit the merge into your local repository. From the revision history viewer (*TortoiseHg* → *Workbench* or **thg log**) open the context menu over the changeset which you want to merge and select **Merge with local...**. Finally, in the merge dialog, press **Merge** and then **Commit**.
4. Ensure your merged work still builds and passes your extensive test suite.
5. Push your changes to the group repository, *TortoiseHg* → *Workbench* or **thg log**, select the path to group repository and then click the **Push** button.

Which may sound complicated, but is easier than it sounds.

Note: Merges can be safely restarted if necessary.

Mercurial makes collaboration easy, fast, and productive. Learn more at Mercurial's wiki.

CHAPTER

FIVE

TORTOISEHG IN DAILY USE

5.1 Common Features

These features are common to many TortoiseHg tools, so we document them here just once.

5.1.1 Visual Diffs

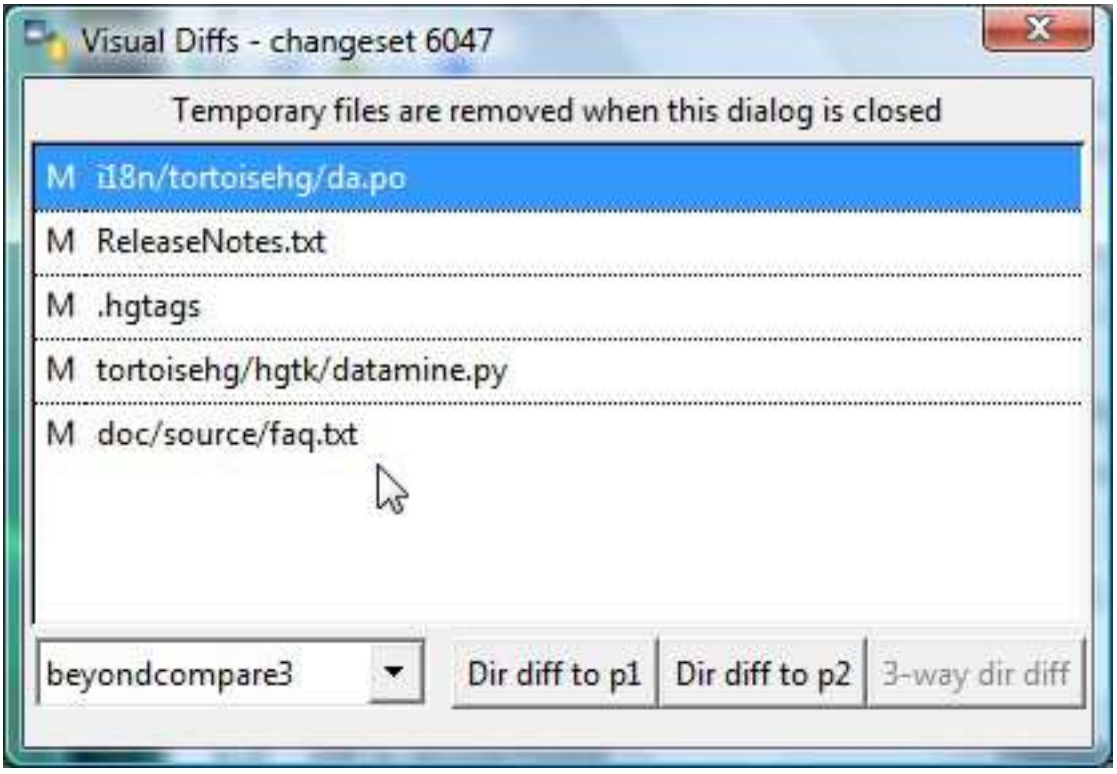


Figure 5.1: Visual Diff Window

In TortoiseHg 1.0, the visual (external) diff infrastructure was refactored. The new system uses tool descriptions in `mergetools.rc` to detect most common diff tools on your computer (including KDiff3, which ships in our installer) and select the best available tool.

If the user has selected a merge tool (*TortoiseHg → Three-way Merge Tool*), that tool will also be used to perform visual diffs, bypassing the tool selection process. However the user can still select a separate tool (*TortoiseHg → Visual Diff Tool*) for visual diffs if they chose.

The merge tool configuration file contains optimal command lines for each tool, so no further configuration is required by the user. They only need to select the tools they wish use, or accept the defaults.

The visual diff system will use any existing extdiff configuration it finds. Since extdiff did not support three way diff arguments until very recently and still does not support label arguments, you will likely have a better experience by disabling or deleting any extdiff configuration you may have.

The visual diff system will directly use the selected diff tool unless the action you are attempting requires the use of the TortoiseHg visual diff window. The list of conditions includes:

1. The selection of files being compared require multiple tools
2. The selected tool forks detached background processes
3. The selected tool does not support the required directory diffs
4. The selected tool does not support three way comparisons
5. The file changes include renames or copies

When the visual diff window is used, the temporary files are cleaned up when the dialog is closed. Thus it should be left open until you close all of your diff tool instances. When your diff tool is launched directly, the temporary files are deleted when your tool exits.

If your diff tool is launched directly to compare a working copy file, it will directly diff against the working file so you may modify it from within the diff tool. If you are comparing multiple files, the visual diff system will make a snapshot of the working copy files and track their initial sizes and timestamps. When your diff tool exits, the system compares the sizes and timestamps and copies modified files back over the original working copies. In this way, you can still modify your working copy files from your visual diff tool even when performing directory comparisons.

When the visual diff window is used to compare working copy files, it always directly diffs against the working copy files since it always operates on a single file at a time.

Note: The *TortoiseHg → Skip Diff Window* configurable has been removed because it is now redundant.

Adding Tools

If you have a visual diff tool installed that is not supported by TortoiseHg, you can create a tool configuration for it in your user `Mercurial.ini` file. See Mercurial's documentation on how to configure your tool for use in file merges. When that is complete, you can add the extra keys used by TortoiseHg for visual diff:

```
diffargs:  the arguments to use for two-way file comparisons
diff3args: the arguments to use for three-way file comparisons
dirdiff:   this tool supports two-way directory comparisons
dir3diff:  this tool supports three-way directory comparisons
```

When building command line arguments, you can use the following variables:

```
$parent1:  the file or directory from the first parent revision
$parent2:  the file or directory from the second parent revision
$child:    the file or directory from the revision being compared
$ancestor: the file or directory from the ancestor of a merge
$parent:   a synonym for $parent1

$plabel1:  a symbolic name for the first parent revision
$plabel2:  a symbolic name for the second parent revision
```

```
$clabel:   a symbolic name for the revision being compared
$alabel:   a symbolic name for the ancestor revision
```

Obviously, $parent2 and $ancestor are only meaningful when used in three way diff arguments, for viewing merge changesets. If your diff tool cannot use the ancestor revision in any productive way, it is safe to leave it out of the diff3args command line.

Note: On Windows, the *executable* parameter can use environment variables using the syntax ${ProgramFiles}

If unconfigured, the default value of **diffargs** is '$parent $child'. The default value of **diff3args** is "", indicating the visual diff tool cannot perform three way comparisons.

If you create a new visual diff tool configuration, or improve upon an existing configuration, please email it to our development mailing list so it may be incorporated in a future release.

Word Diffs

The TortoiseHg Windows installers now include TortoiseSVN's scripts for comparing (and sometimes merging) many binary document formats. These are configured in the site-wide `mergepatterns.rc` as handlers for each binary format's common file extensions, so no user intervention is required.

In order to support file extension based tool selection, TortoiseHg has added support for a **[diff-patterns]** section equivalent to Mercurial's merge-patterns section.

5.2 Windows Explorer Integration

5.2.1 Context Menus

TortoiseHg commands may be accessed via the context menu of Explorer windows and other applications which use the standard File/Open dialogs. Here is the context menu for a revisioned folder:

And here is the context menu for selected files or folders:

TortoiseHg provides dialogs for the most regularly used Mercurial commands. Less frequently used and newly added Mercurial commands may be accessed from the CLI (command line interface) through `cmd.exe` on Windows.

5.2.2 Overlay Icons

TortoiseHg provides visual representation of the file status via overlay icons in the MS-Explorer windows. This is similar to those that found on other Tortoise client, such as TortoiseCVS and TortoiseSVN.

TortoiseHg shares the overlay icons with TortoiseSVN (version 1.5.x or later) and the other "Tortoise" projects via the use of TortoiseOverlays (another project created by TortoiseSVN team).

The context menu has an **Update Icons** option which forces TortoiseHg to refresh the icons in the currently browsed repository or directory of repositories. The taskbar icon will turn green and the directory icons will turn into question marks while this refresh is in progress.

5.2.3 Shell Configuration

The overlay handler and context menus are configurable. From any folder background (even the desktop), right click and select *TortoiseHg → Explorer Extension Settings*. This opens the TortoiseHg Shell Configuration dialog.

On the tab "Context Menu" you can promote individual menu options to the top level menu.

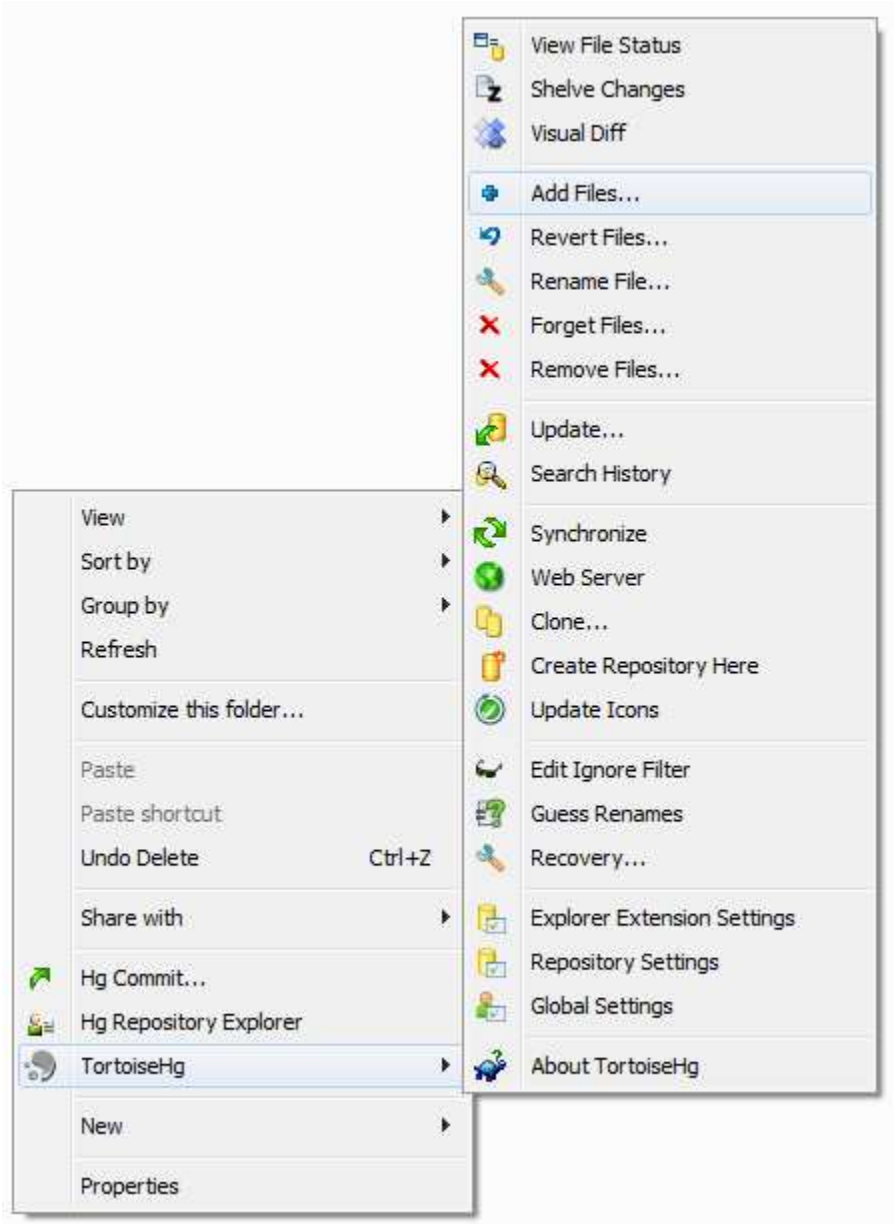


Figure 5.2: Context menu for a folder under Mercurial revision control

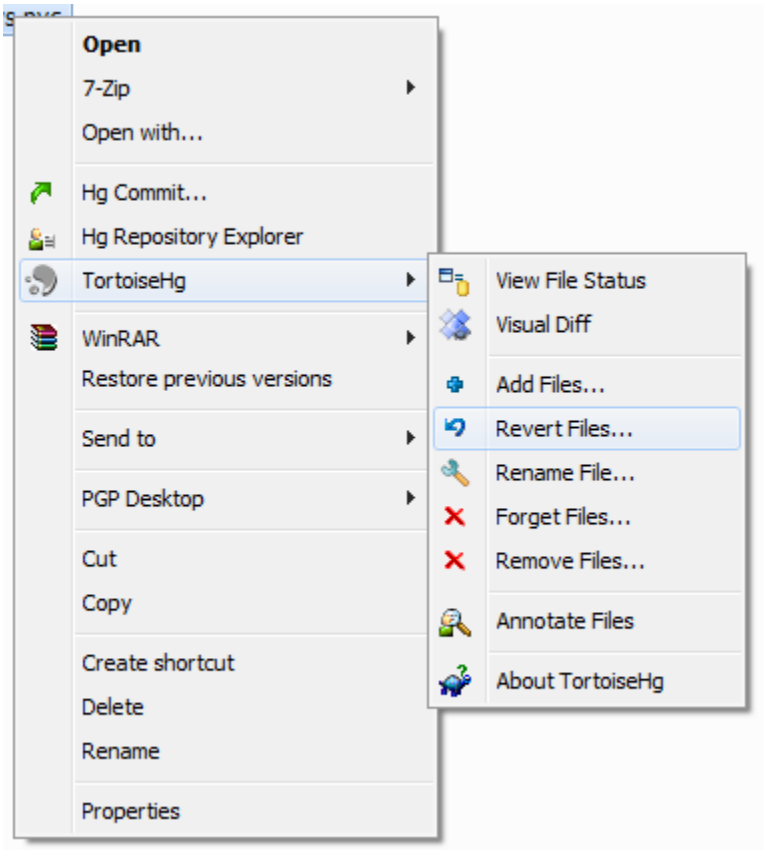


Figure 5.3: Context menu for file or folder selection

Figure 5.4: Overlay icons in Icons view (XP)

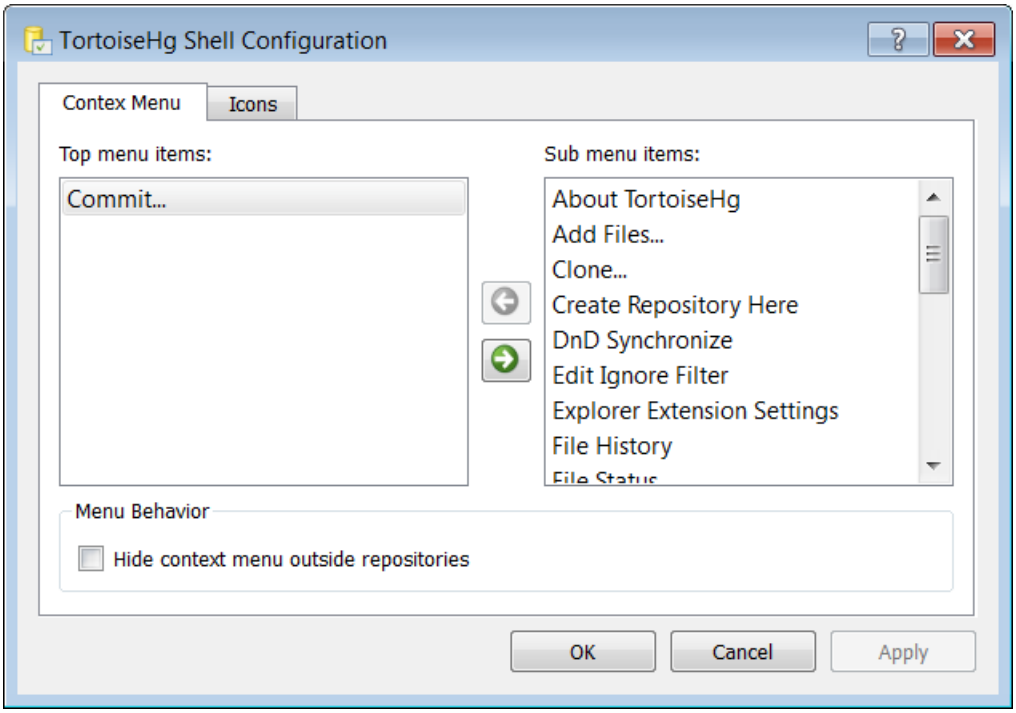


Figure 5.5: Shell Configuration Dialog, Context Menu tab

On the "Icons" tab you configure settings related to the overlay icons and the icon of the "Overlay Icons Server" in the taskbar (in the notification area of Windows).

Enable overlays: If checked, overlay icons are shown on folders and files in the working directory (woking copy) of Mercurial repositories. (Default: checked)

Local disks only: If checked, overlay icons are only shown for volumes on local disks, not on network shares. Scanning for Mercurial repositories over the network may result in high latency in the user interface of explorer. Check this option if browsing network shares becomes too slow and/or you do not need overlay icons on non-local volumes. (Default: not checked)

Enabled Overlay Handlers: These (per user) settings provide the possibility to disable overlay icon handlers in the shared TortoiseOverlays component. The TortoiseOverlays component is shared by all Tortoises (TortoiseHg, TortoiseSVN, etc), with the goal to avoid registering too many icon slots, by using a common set of icons slots for all Tortoises (thus using the same set of icons for all Tortoises). The total number of overlay slots available on Windows is fairly limited and depends on the exact Windows version. For example, on a pristine install of Windows 7, there are only 8 free overlay handler slots available. This section allows to disable certain non-essential overlay handlers, to reduce icon handler slot consumption by the TortoiseOverlays component. Unchecking handlers in this section increases the chances that important handlers like "Normal" (green checkmark) or "Modifed" (red exclamation mark) will still get an icon slot, even if there are too many handlers registered on a computer. Unchecking handlers that are not used by TortoiseHg (that is: Locked, Readonly, Ignored, Deleted) is highly recommended, if you know that no other Tortoises (e.g. TortoiseSVN) uses them. Make sure the "Added" and "Unversioned" handlers are enabled, as these are used by TortoiseHg. (Default: all checked)

> **Warning:** The "Enabled Overlay Handlers" settings affect all Tortoises for a user. A logoff/login is required to make changes in that section effective.

Taskbar: Checkmark "Show Icon" to show the icon of the Overlay Icon Server in the taskbar in the notification area. "Highlight Icon" highlights that icon using a light green color while the icon server is busy updating cache files

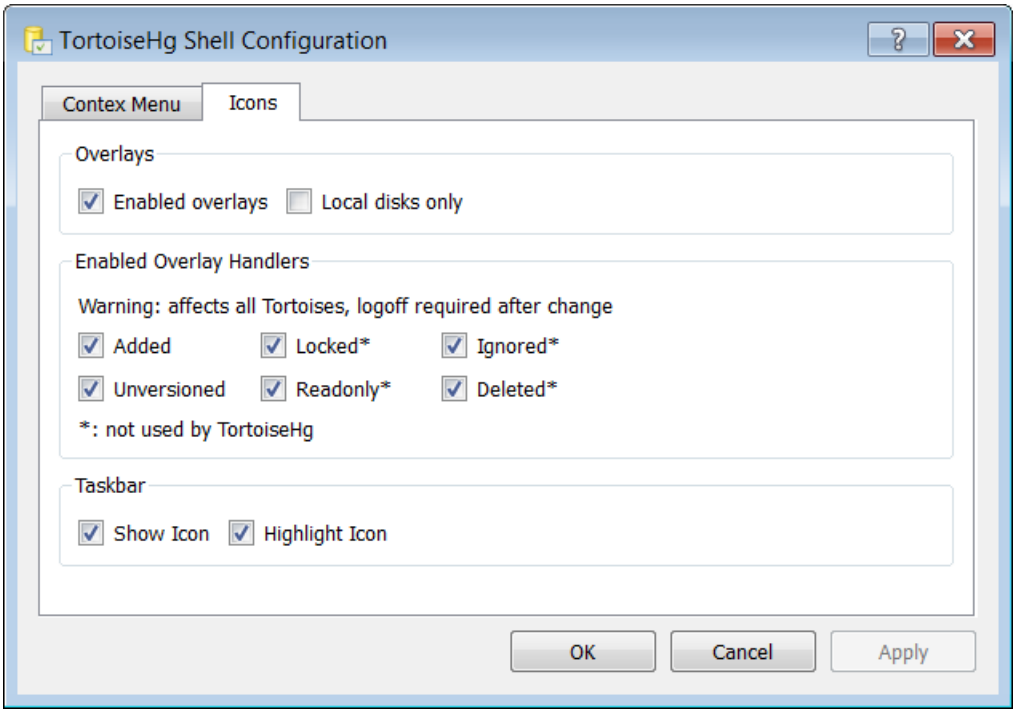


Figure 5.6: Shell Configuration Dialog, Icons tab

in the repository (files `.hg\dirstate` and `.hg\thgstatus`). (Default: both checked)

One can selectively disable overlay icons in a specific repository by editing the `.hg\thgstatus` file inside the repository and replacing its contents with a single line containing:

```
@@noicons
```

5.3 GNOME desktop integration

TortoiseHg also provides shell integration with the GNOME desktop via a nautilus-python plugin. If you have installed TortoiseHg from a distribution package, the odds are that this extension is already configured. If not, please consult our Wiki for instructions on how to enable this feature.

While the nautilus extension does not have its own GUI for managing the overlays and context menus, it does support command promotion into the top menu. It requires you to edit your `~/.hgrc` file and add lines like these:

```
[tortoisehg]
promoteditems = commit, log, synch
```

5.4 Workbench

The Workbench is the primary TortoiseHg application. It allows you to browse your local repositories, make commits, perform searches, synchronize with other repositories, and perform various maintenance tasks. Nearly every Mercurial feature is accessible from the Workbench.

Workbench Main Widgets are:

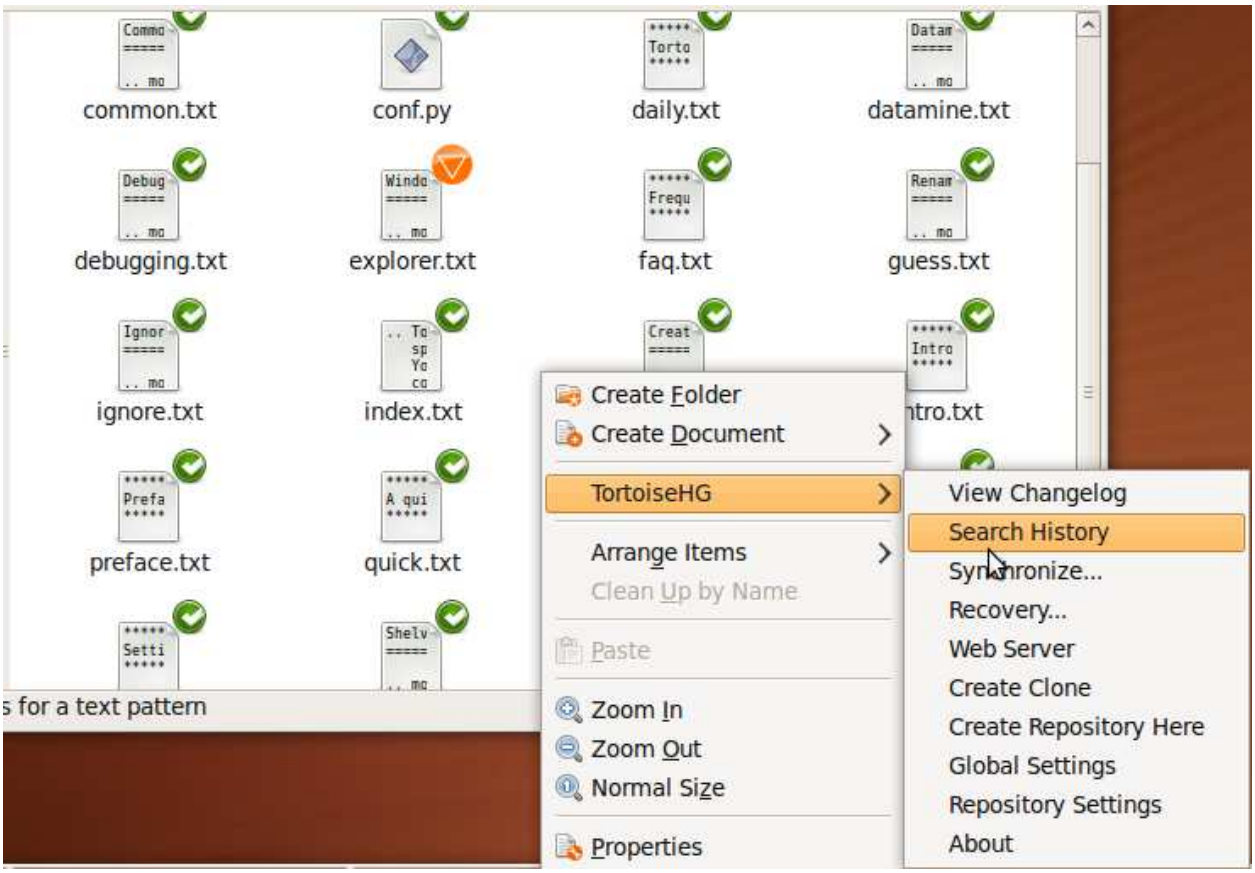


Figure 5.7: GNOME/Nautilus screenshot

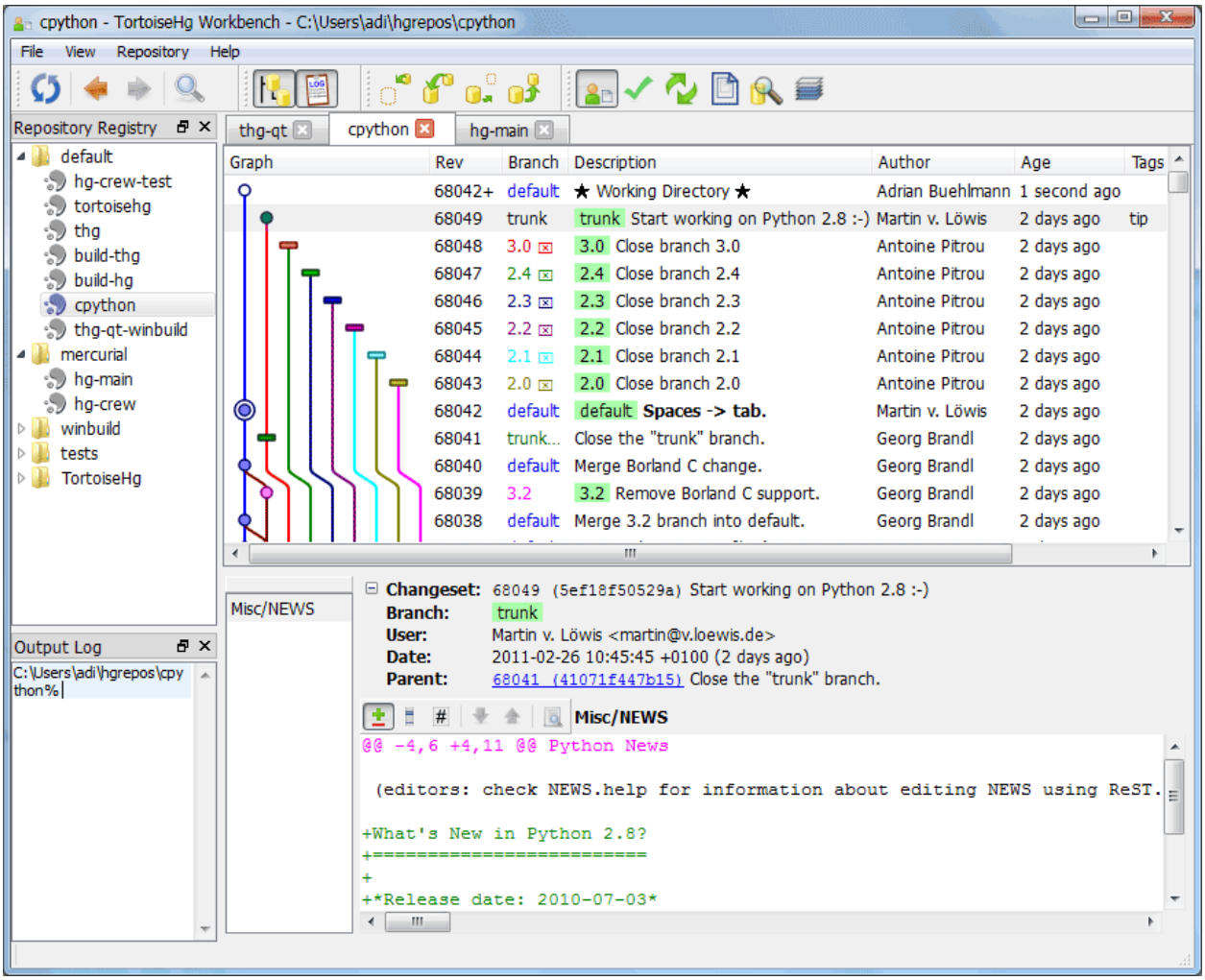


Figure 5.8: Workbench dialog.

Revision History View A tabbed widget to view multiple repositories at once. The different columns show general information about each changeset in the graphlog. You can configure the columns to show from the menu via *View* → *Choose Log Columns...*, and there you can reorder the columns too. This is the main or central widget of the Workbench application.

Repository Registry This widget, by default shown on the left, allows to manage multiple repositories from the Workbench. You can show/hide it via *View* → *Show Repo Registry* or with the corresponding button in the Dock Toolbar. It's also a dockable widget. The *View* → *Show Paths* menu option allows to not only view the names of the repositories but also their path in a second column.

Output Log This dockable widget, which can be shown/hidden with *View* → *Show Output Log*, gives the user information about the Mercurial commands that were executed during the current session. You can also use it as a commandline by typing Mercurial commands at its prompt directly. It shows any error messages when appropriate. Content is wiped when the Workbench is closed.

Task Tabs The lower right part of the Workbench is occupied by a stack of widget where you can perform various frequent tasks. It is a tabbed widget. See further for more detail about each one.

5.4.1 Workbench Menus

The Workbench has a menu bar for accessing tool functions and for launching other tools.

File Handle repositories and settings.

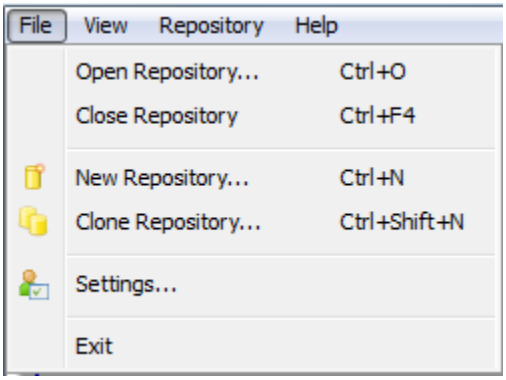


Figure 5.9: File Menu

View Manage the visibility of various parts of the Workbench.

Repository Perform special actions on the active repository.

Help About shows TortoiseHg version info.

5.4.2 Edit Toolbar

Moving around in the revision history. All the buttons work on the current repository.

Refresh Reload the revision history of the current repository.

Back Go back to the previously selected revision.

Forward Go forward to the next revision in your selection history or most recent revision set query.

Filter toolbar Show and activate the Filter Toolbar at the top of the revision graph.

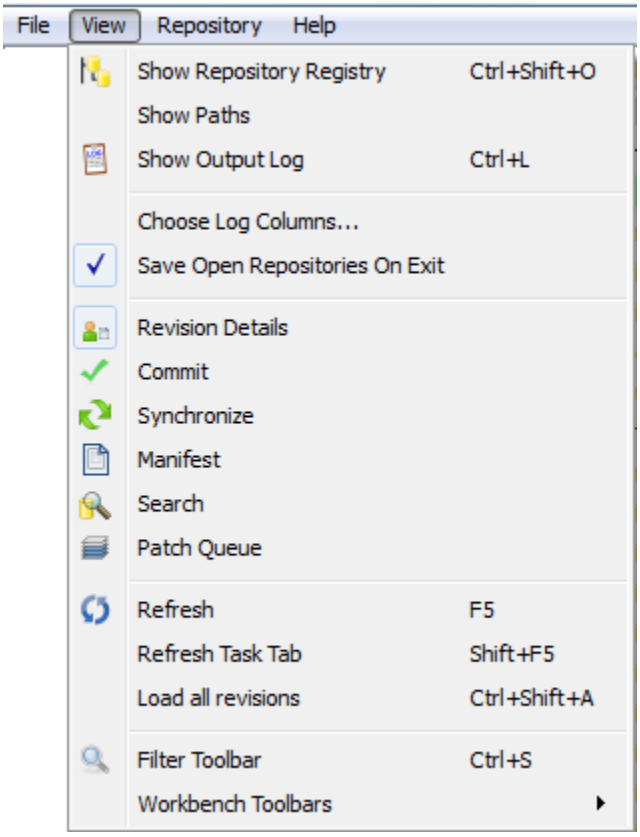


Figure 5.10: View menu

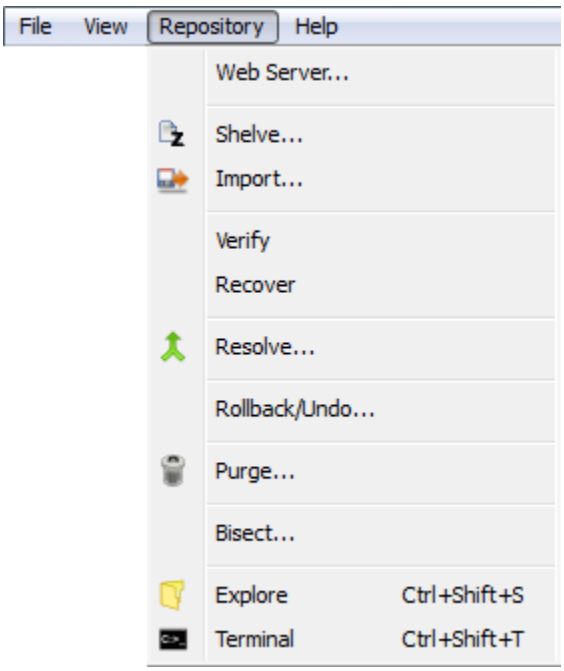


Figure 5.11: Repository menu

Figure 5.12: Edit toolbar

5.4.3 Dock Toolbar

Figure 5.13: Dock toolbar

Show or hide some main widgets in the Workbench.

Show Repository Registry Show/hide the Repository Registry widget.

Show Output Log Show/hide the Output Log widget.

5.4.4 Sync Toolbar

Figure 5.14: Sync toolbar

Synchronize your repository with other repositories.

Incoming Download incoming changesets from the remote repository, store then in a temporary bundle file, then enter bundle preview mode with the incoming changes applied. Incoming changesets will be shown as normal, while others will be shown grayed in the revision graph. The buttons **Accept** and **Reject** are then shown at the top of the revision graph.

Pull Pull incoming changesets from the remote repository, then apply after-pull effect (update, fetch, or rebase).

Outgoing Determine outgoing changesets that would be pushed to the remote repository. Outgoing changesets will be shown as normal, while others will be shown grayed in the revision graph.

Push Push outgoing changesets to the remote repository.

5.4.5 Task Toolbar

Work with the various task tabs.

Revision Details Shows information about the current revision : files added, removed, renamed, or modified, file contents, changeset info. See Revision Details for more detail.

Commit Here you can add new files, and do your commits. See Commit for more detail.

Synchronize Gives you full control about how you let your repositories communicate with any other repository. See Synchronize for more detail.

Manifest Shows information about the complete content of the repository as it was for the current revision. Here you can view the raw content of files or an Annotate view. There is a context menu on the filelist to do further digging into the history data. You can even compare a file between different revision from there. See Manifest for more detail.

Figure 5.15: Task toolbar

Search For performing text searches through file content.

Patch Queue This widget implements the MQ extension functionality. See Patch Queue for more detail.

There is some relation between the revision or patch selected in the graph pane, and the task tabs.

- Clicking on the Working Directory automatically switches to the Commit task tab.
- Clicking on any revision other than the Working Directory switches to the Revision Details task tab.

You can overrule this standard behaviour by doing an `ALT-Click` for making your selection. This preserves the current task tab, no matter what revision or patch you select. Cursor selection movements also do not switch task tabs.

5.4.6 Filter Toolbar

Figure 5.16: Filter features for the Workbench.

The filter bar allows one to quickly filter the changesets panel. It is based on the Revision Sets feature of Mercurial. See hg.1.html#revsets for details on how to specify revision sets. The toolbar can be toggled with `Ctrl-S`. Parts from left to right:

Clear Clears the search lineedit. Essentially disables all filters.

Filter entry Here you can type a filtering condition. The widget is a combobox, holding a history of previous filtering conditions.

Trigger Applies the condition set by the filter.

Open Opens the RevSet dialog. There you can select and/or enter your condition in a combined way via point-and-click and by typing.

Delete Deletes the selected query.

Toggle filter Applies the filter condition by sowing changesets that don't conform to it in a color suggesting insensitiveness, so the selected ones stand out more.

Branch options A few options for showing branches. See *Repo Settings* → *Workbench* → *Dead Branches* for a method to prune names from this combo box.

Branches combo A combo box with the list of named branches in your repository.

Custom Filter Combo Finally there is a combo box that selects among the various filter types that can be manually specified.

If the repository tab is previewing incoming changesets, a pair of buttons are prepended to the start of the filter bar:

Accept Accept (pull) the changesets from the previewed bundle. This button is only visible when previewing a changeset bundle. The after-pull effect is respected after pulling from a bundle.

Reject Reject the changesets from the previewed bundle and exit preview mode. This button is only visible when previewing a changeset bundle.

The Workbench will attempt to lookup the entered search phrase in the repository to see if it matches a tag, bookmark, branch name, changeset hash, or revision number. If no changeset match is found, the Workbench checks if the search phrase has any parentheses. If no parentheses are found, the Workbench assumes the search is a keyword and performs a **keyword()** revision set search. If parentheses are found, the Workbench assumes the search phrase is a revision set specification and attempts to resolve the set.

If you need to perform a keyword search that includes parentheses, use **keyword("PHRASE(FOO)")**.

5.4.7 Revision Graph Details

The graph column shows the child-parent relationships between revisions in your repository history. This column auto-sizes for as many lines of ancestry that are required to visualize the revisions you have loaded. The column has an initial hard-limit width to prevent some degenerative cases from breaking the viewer, but can be resized after refreshes.

5.4.8 Performance Implications

There are some Workbench features that could have performance implications in large repositories.

View* → *Choose Log columns... Enabling the **Changes** column can be expensive to calculate on repositories with large working copies, causing both refreshes and scrolling to be slow.

View* → *Load all Normally, when the user scrolls through the history, chunks of changesets are read as you scroll. This menu choice allows you to have the Workbench read all the changesets from the repository, probably allowing smoother moving through the history.

5.4.9 Revision Context Menus

Right-clicking on revisions in the graph pane brings up a different context menu when one, two, or more revisions are selected. Context menus can also differ according to the type of revision(s) (working dir, regular revision, (un)applied mq patch). Here we give a list of all existing context menu entries.

Right-clicking on a selection of revisions in the (top) graph pane will bring up the revision context menu.

With only one revision selected: Update... Update your working directory to this revision. Opens the TortoiseHg update dialog with this revision selected.

Visual diff... Open this change in your visual diff tool.

Diff to local... Display changes (visual diff) between this revision and your current working directory.

Browse at rev... Brings up the Manifest window with the content of all files in the repo at the selected revision.

Merge with local... Merge the selected changeset with the Working Dir. Opens the TortoiseHg merge dialog with this revision selected.

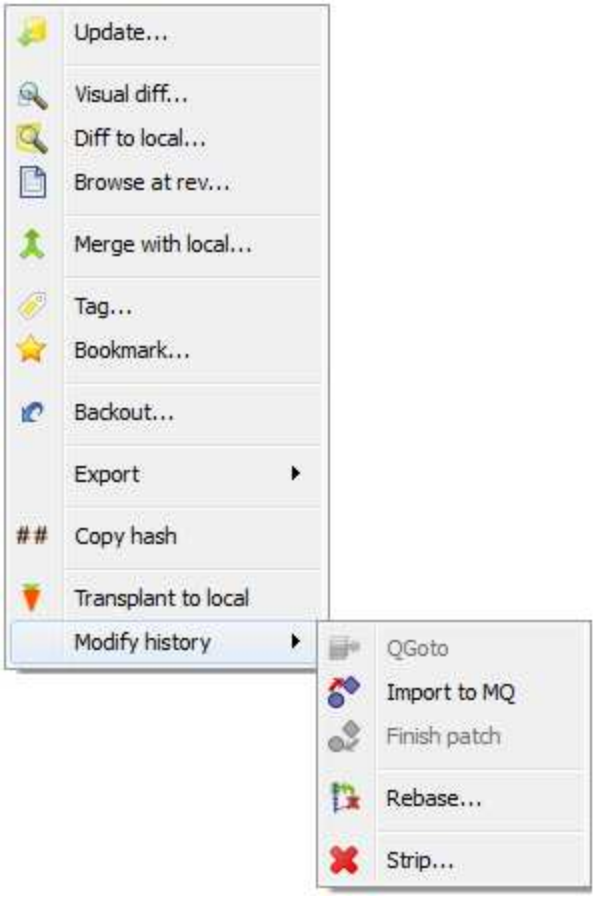
Tag... Allows to manage tags to the selected revision.

Bookmark... Allows to manage bookmarks for the selected revision. *This option requires the boomarks extension to be enabled.*

Backout... Create a backout changeset for selected revision.

Export Export patch Generate a patch file containing this revision's changes.

Email patch... Send this revision's changes to email recipient. Opens the TortoiseHg email dialog with this revision selected.


Update...
Visual diff...
Diff to local...
Browse at rev...
Merge with local...
Tag...
Bookmark...
Backout...
Export
Copy hash
Transplant to local
Modify history
QGoto
Import to MQ
Finish patch
Rebase...
Strip...

Archive... Open the archive dialog for this revision, allowing user to generate a backup copy of the repository at that revision.

Copy patch todo *Only visible when MQ is enabled.*

Copy hash Copy current revision's full hash to the clipboard.

Modify history **QGoto** Push/pop patches upto this one *Only visible when MQ is enabled*

Import to MQ Import selected revision into the current patch queue. Only valid for qbase or checked out head revision. *Only visible when MQ is enabled*

Finish patch Transforms the MQ patch into a regular changeset. *Only visible when MQ is enabled*

Rebase... Move the selected revision and all of its descendants onto the current working parent revision. *Only visible when rebase is enabled*

Strip... Remove the selected revision and all of its descendants from the repository [1] *Only visible when MQ is enabled*

With two revisions selected: **Visual diff...** Open this change in your visual diff tool.

Export selected Creates a patch file for each changeset in selected range.

Email selected... Opens email dialog with range of changesets.

Export DAG range Creates a patch file for each changeset in selected range.

Email DAG range... Opens email dialog with range of changesets.

Bisect - Good, Bad... todo See bisect section below.

Bisect - Bad, Good... todo See bisect section below.

Compress history... Brings up a dialog where you can compress the youngest changeset into the older one.

With more than two revisions selected: **Export selected** Creates a patch file for each changeset in selected range.

Email selected... Opens email dialog with range of changesets.

5.4.10 File Context Menus

Right-clicking on filenames in the file list pane (bottom left of the RevDetails and Manifest task tabs) will bring up a context menu for the selected file:

Visual Diff Open this revision of the file in your visual diff tool.

Visual Diff to Local Visualize differences between this revision and your checked out version.

View at Revision Open this revision of the file in your visual editor [2].

Edit Local Open the checked out version of the file in your visual editor [2].

Revert to Revision Checkout this specific revision of this file [3].

File History Show revisions that modified this file [4].

Compare file revisions Brings up a new dialog where you can compare any revision of the file with any other revision in the history.

Right-clicking on filenames in the file list pane of the Commit task tab will bring up a different context menu for the selected file:

[1] The strip command will store the stripped revisions in a bundle file that can later be reapplied. See also EditingHistory.
[2] *Global Settings* → *TortoiseHg* → *Visual Editor*
[3] The new contents will appear as local changes and must be committed.
[4] Does not show revisions where a file was deleted, as this is only a manifest change, it does not modify the file's history.

Edit Open this revision of the file in your visual diff tool.

Add Add this file to the repository for versioning.

Detect Renames... Brings up a dialog where you can try to detect renamed files.

Ignore Adds the selected file to the .hgignore content.

Delete unversioned Deletes unversioned files from disk.

5.4.11 Message Parsing

The changeset display pane will detect and underline changeset hashes, HTTP(s) URLs, and bug report identifiers inside changeset messages. These underlined phrases are clickable links.

Every word-boundary delimited string of 12 or 40 characters from the range [0-9a-f] is considered a changeset link. Clicking on it in the repository explorer will jump to the given changeset if possible.

HTTP and HTTPS URLs are similarly turned into clickable links which are opened in your default web browser.

Issue tracker links are enabled when configured in the tortoisehg section of your configuration files. Since only a single issue tracker can be configured at a time, it is typically configured in the repository's `.hg/hgrc` file. There are two keys: issue.regex and issue.link. The first defines the regex to match when picking up issue numbers, while the second defines the command to run when an issue number is recognized.

You may include groups in issue.regex, and corresponding {n} tokens in issue.link (where n is a non-negative integer). {0} refers to the entire string matched by issue.regex, while {1} refers to the first group and so on. If no {n} tokens are found in issue.link, the entire matched string is appended instead.

Examples:

```
BitBucket:
issue.regex = #(\d+)\b
issue.link = http://bitbucket.org/<your project and repo>/issue/{1}/

Mercurial:
issue.regex = \bissue\d+\b
issue.link = http://mercurial.selenic.com/bts/
```

5.4.12 Output Log Console

The console built into the Workbench Output Log dock widget can run Mercurial (hg) commands, TortoiseHg (thg) commands, a couple special commands, and limited shell commands. Commands are always executed in the root of the current repository. The prompt is updated to keep you aware of the context.

If the command line begins with 'hg', the Mercurial command is run in TortoiseHg's execution environment; meaning output is sent to the log widget and input requests are handled by dialog windows.

If the command line begins with 'thg', the requested command is run in a new window but in the same process. For instance 'thg ci' will open a new commit tool window for the current repository.

It the command is 'clear' (or 'cls'), the output log contents are erased.

If the command is 'exit', the output log window is closed.

Otherwise, the command line is forwarded to your platform's default command shell with a limited execution context. There is no stdin while stdout and stderr are piped to the output log.

5.4.13 Keyboard navigation

Ctrl-P Zoom to the working directory parent revision

Ctrl-D Display visual diffs for selected changeset or file

Ctrl-S Toggle revision set / filter toolbar

See also KeySequences on the wiki pages.

5.4.14 Configurables

The Workbench has a few configurable options that can be set in the TortoiseHg Settings dialog on the Workbench tab.

Author coloring If true, each author's changeset will be given a unique color

Long Summary Concatenate commit message lines until 80 chars are reached

Graph batch limit Number of revisions to read in each batch load

Dead Branches Comma separated list of branch names that should be ignored when building a list of branch names for a repository.

Branch Colors Space separated list of branch names and colors on the form branch:#XXXXXX. Spaces and colons in the branch name must be escaped using a backslash (\). Likewise some other characters can be escaped in this way, e.g. \u0040 will be decoded to the @ character, and \n to a linefeed.

Hide Tags Space separated list of tags that will not be shown. Useful example: Specify "qbase qparent qtip" to hide the standard tags inserted by the Mercurial Queues Extension.

The exact colors given to particular users can be configured by adding lines like these to your `Mercurial.ini` file:

```
[tortoisehg]
authorcolor.USERNAME = color
```

The Workbench also respects the following settings on the TortoiseHg tab:

Tab Width Number of spaces to expand tabs in diffs

Max Diff Size Maximum size of file to be diffed

5.4.15 From command line

The Workbench can be started from command line

```
thg log [OPTIONS] [FILE]

aliases: history, explorer, workbench

workbench application

use "thg -v help log" to show global options
```

5.5 Create a new repository

To create a new repository into an existing directory (project) you have to run the init dialog. From the explorer context menu select *TortoiseHg...* → *Create Repository Here* over the directory, or, within the folder, type **thg init**.

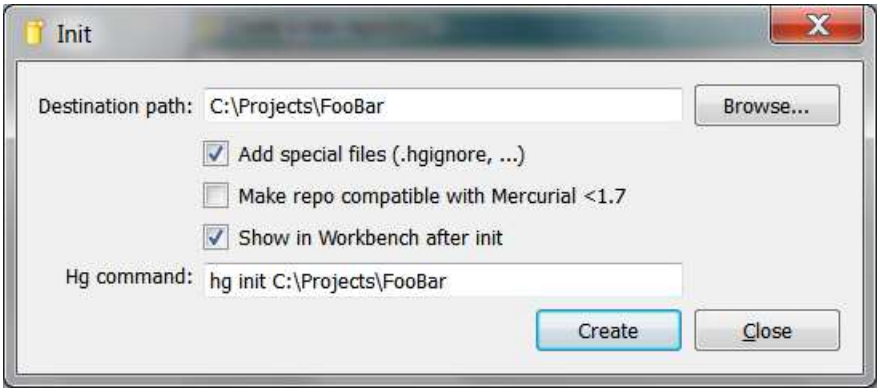


Figure 5.17: Repository Init Dialog

Destination Is the directory where the repository will be created. It is always filled with the current directory, so if you launch the dialog from the right directory there is no reason to change it.

Add special files (.hgignore, ...) If selected TortoiseHg creates an empty `.hgignore` file in the working directory.

Make repo compatible with Mercurial <1.7 If selected TortoiseHg creates an older format Mercurial repository. Do not check unless you have a strong reason to do, and you know what you are doing.

Show in Workbench after init When the repository was successfully created, it is added to the RepoRegistry, and opened in a new tab the Workbench.

Hg command This field displays the command that will be executed by the dialog.

Creating a new repository means create a subdirectory called `.hg`. In this subdirectory Mercurial keeps all its versioning information.

> **Warning:** It is dangerous to manually edit the files in `.hg` directory, repository corruption can occur. `.hg/hgrc` is perhaps the only exception to this rule.

5.5.1 From command line

The init tool can be started from command line

```
thg init
```

The syntax is

```
thg init [DEST]
```

where [DEST] is the path to destination folder.

5.6 Clone a repository

To clone a repository you have to run the clone dialog. From the explorer context menu select *TortoiseHg...* → *Clone a repository* or type **thg clone**.

Source It is the path (or URL) of the repository that will be cloned. Use the **Browse...** to choose a local folder.

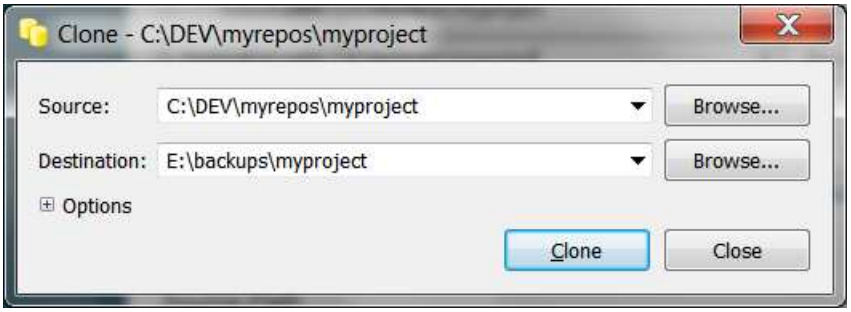


Figure 5.18: Clone Dialog

Destination It is the path of destination directory, a folder with the same name of source repository will be created within this directory.

Under the **Options** expander you will find:

Clone To Revision You can limit the clone up to this revision. Even the tags created after this revision will not be imported.

Do not update the new working directory If checked, after the clone the working directory will be empty. It is useful when you have to clone a repository with the purpose of central repository, or backup, where you have only, in the future, to *push* and *pull*.

Use pull protocol to copy metadata When the source and destination are on the same filesystem, Mercurial tries to use hardlinks. Some filesystems, such as AFS implement hardlink incorrectly, but do not report errors. Use this option to avoid hardlinks.

Use uncompressed transfer To use uncompressed transfer (fast over LAN).

Include patch queue To also clone an MQ patch repository along with the main repository.

Use proxy server To use the proxy server configured in *TortoiseHg... → Global Settings → Proxy*. This is enabled only if a proxy is configured.

Remote command Specify a Mercurial command to run on the remote side.

5.6.1 From command line

The clone tool can be started from command line

```
thg clone
```

The syntax is

```
thg clone [SOURCE] [DEST]
```

where [SOURCE] and [DEST] are, the paths of source repository and destination folder.

5.7 Commit

The commit tool is second most commonly used application after the Workbench. Not only can the commit tool commit your changes, but it can also examine the state of your working directory and perform most routine maintenance tasks (add new files, detect renames, manage the ignore filter, etc).

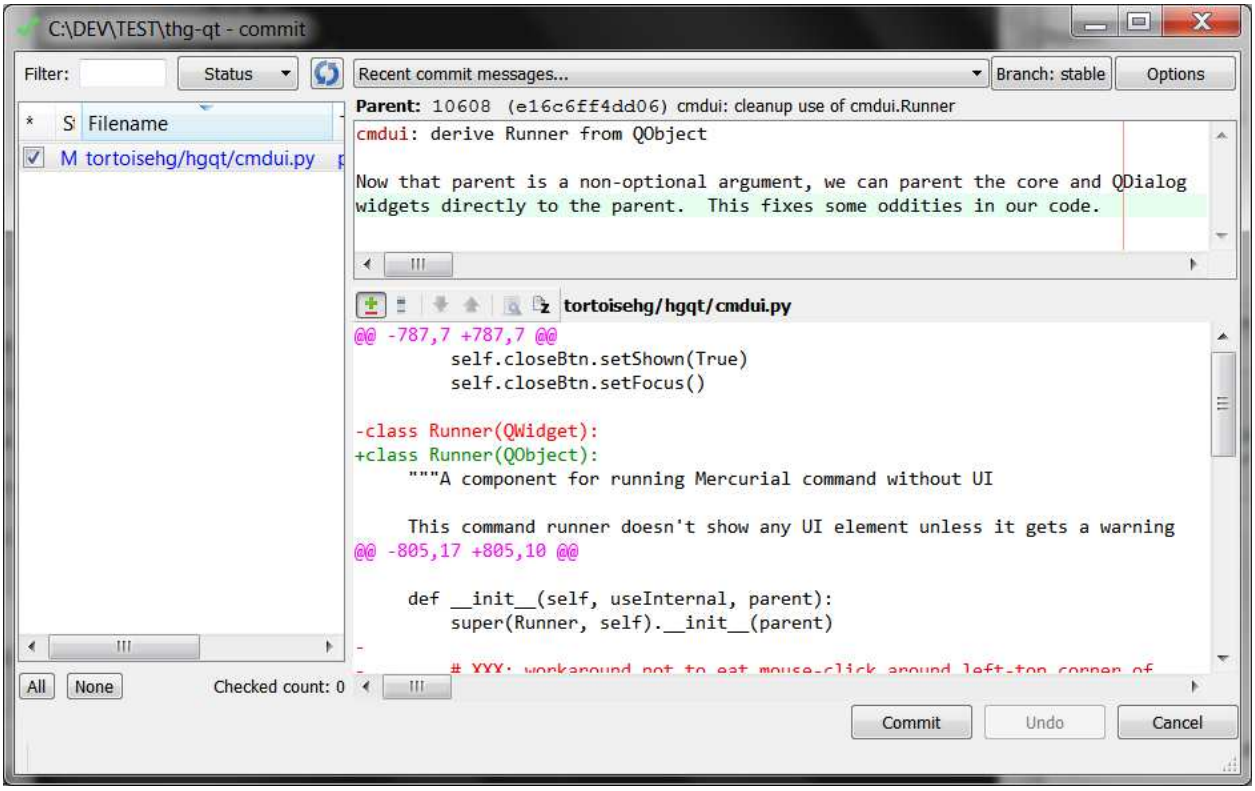


Figure 5.19: Commit dialog

5.7.1 Features

Enumerating the toolbar buttons:

Branch dialog Shows the current branch name of the working directory. Normally this is informational only, but pressing this button opens up a branch maintenance dialog. Do not use this feature unless you understand Mercurial's named branches.

Recent Commit Messages A drop-down list of the 10 most recent commit messages. The the drop-down list is filled the first time it is opened.

Commit Commit selected diffs in checked files.

Undo Undo (rollback) last immediate commit. Your commit message will be available in the message history, so you can easily repeat the commit if necessary.

The file list has four columns:

1. A checkbox that indicates whether the file is selected for an operation. The toolbar buttons only operate on checked files. "Partially" selected files have a special check state. This column header is checkable, it will toggle the file selection states.
2. The **st** column holds the status of the file, defined by Mercurial's status command, one of 'MARD?IC'. A status of 'S' indicates a dirty subrepository that needs to be committed.
3. The **ms** column holds the merge state of the file, defined by Mercurial's resolve command, one of ' RU'. See the merge section below.
4. The canonical path of the file relative to the repository root

Note: If the commit tool was started with a file pattern or selection, a button will appear at the bottom of the file list that can clear the file pattern and give you an unfiltered view of the entire working directory.

The **Status** button has a menu with checkable options that toggle the display of the various classes of files {modified, added, removed, deleted, unknown, clean, ignored}.

Removed means a revisioned file has been marked as removed. *Deleted* means a revisioned file is missing but Mercurial has not been told to quit tracking that file. For instance, if you rename a revisioned file in Explorer, the original filename will show up as deleted and the new filename will show up as unknown. By right-clicking on the new filename you can bring up the rename guessing dialog which can discover the rename by comparing file contents and mark the old file as removed and the new file as added while recording the whole operation as a rename.

Unknown files are not tracked by Mercurial, but they also do not match any ignore filters you have configured. Unknown files are shown by default because they are usually files that need to be added to revision control. It is recommended that you keep your ignore filters up to date to ensure that is the case. The context menu of unknown files has an option open the ignore pattern tool.

Clean files are tracked files that have not been modified, while *Ignored* files are untracked files that match a configured ignore pattern. Neither of those file types are shown by default, unless a the user includes such a file in a selection (explorer) or provides the file name on the command line.

5.7.2 Keyboard navigation

`Ctrl-Enter` Trigger the commit

`Ctrl-E` Reflow the paragraph currently under the cursor. You must configure a message format policy for this shortcut to work.

5.7.3 File Context Menus

When right clicking on files in the file list, you will get a context menu of commands that are applicable to the selected files.

For unknown **?** files, the context menu will allow you to detect renames (if you think the unknown file is a copy or rename of a revisioned file) or to configure the repository's ignore filter (if the unknown file should never be revisioned and you want Mercurial to ignore it).

5.7.4 Merging

The commit tool has a special mode when it is opened in a repository that is in a merged state (either a merge is in progress, or an update was performed that caused a conflict).

The merge state *ms* column is especially useful in this mode. Files that are marked with *R* are files where Mercurial and/or the user have successfully merged (resolved) changes from both parents. Files that are marked with *U* have unresolved changes. You can use the *Restart Merge* context menu option to restart the merge for those files, or you can use the *edit* context menu option to resolve the conflict by hand. The *Restart Merge* menu option allows you to select the merge tool to use to perform the merge, or even to pick one version or the other unconditionally (internal:local, internal:other). After the conflicts have been manually resolved, you must use the *mark resolved* context menu option to change the file's merge state to *R*.

Mercurial will not allow you to commit a merge if any files have unresolved *U* merge states.

For your reference, *local* is the revision you had checked out when you started the merge and *other* is the revision you merged with.

To undo a failed merge attempt, you must tell Mercurial to remove the second parent from your working directory. This usually means performing a clean update of the first parent. The merge tool has an **Undo** button which does exactly that.

Once you have your working directory back at one parent revision, you may restart the merge process.

5.7.5 Commit Message Pane

The commit message pane has these special context menu options:

Paste Filenames: Paste checked filenames into the commit message at the cursor.

Apply Format: Apply configured message wrap policy to current message.

Configure Format: Opens the settings dialog to the **Commit** tab.

If your project has guidelines for the format of commit messages, you can configure them in the settings tool. The commit tool will enforce your policy at commit time, and you can ask the tool to apply the format to the current message. The **Commit** tab of the settings tool has these two configurables for commit message policy:

Summary Line Length: Maximum length of the commit message summary line. If set, TortoiseHg will draw a line at the specified width.

5.7.6 Subrepositories

A subrepository is a feature introduced in Mercurial 1.3. It allows one Mercurial repository to store references to external Mercurial (or potentially other VCS) repositories, and to include the state of those external repositories in the main repository's history.

TortoiseHg 1.0 introduced rudimentary support for subrepositories, and only in the commit / status tool. When Mercurial considers a subrepo dirty, it will appear in the commit tool as a special entry in the file list with a status of *S*. If a subrepo is included in the file list of a commit, the subrepo is committed along with the other changes, updating the .hgsubstate file in the main repository root.

5.7.7 Configurables

Commit* → *Username Sets username associated with your commits (see *A Quick Start Guide to TortoiseHg*)

Commit* → *Summary Line Length Configures a 'policy' limit for summary lines

***Commit* → *Close After Commit*:** When set to True, the commit tool will close after a successful commit.

And three other features for *advanced* users.

***Commit* → *Push After Commit*:** If configured, the commit tool will try to push to the configured URL or alias after each commit.

***Commit* → *Auto Commit List*:** Comma separated list of files that are automatically included in every commit. Intended for use only as a repository setting.

TortoiseHg* → *Max Diff Size Configures the diff size limit

5.7.8 From command line

The commit tool can be started from command line:

```
thg commit [OPTIONS] [FILE]...

aliases: ci

commit tool

options:

 -u --user  record user as committer
 -d --date  record datecode as commit date

use "thg -v help commit" to show global options
```

For a quick help on the format of date type:

```
hg help dates
```

5.8 Shelve

The shelve tool can move changes between the working directory and shelf patches. If the MQ extension has been enabled, it can also move changes into and out of unapplied patches.

The shelve tool can be launched by the Workbench **Repository –> Shelve** menu option, by a toolbar button on working file viewers, or by **thg shelve**.

Note: We highly recommend setting the patch eol configuration to auto if you use the shelve tool with DOS eoln text files.

5.8.1 Features

The shelve tool has three toolbars. A right and left toolbar for the two side by side panels, and a central toolbar for refresh and creating a new shelf patch. The right and left toolbars are mirrors of each other, offering the same functionality in alternate directions.

The left toolbar has these actions:

Delete selected chunks Remove, or revert, all selected (toggled) chunks in the currently selected file.

Move all files right Move all changes in all files to the patch selected in the right pane.

Move selected file right Move all changes in the currently selected file to the patch selected in the right pane.

Edit selected file If the working directory is being browsed, this button edits the currently selected file. Else it edits the currently viewed patch file.

Move selected chunks right Move all selected (toggled) chunks to the patch selected in the right pane.

The central toolbar has two actions:

Refresh Refreshes the patch drop down lists and the working copy view

New Shelf Creates a new shelf file. You must enter a name, though a reasonable default is provided.

The right toolbar will move changes from the patch selected on the left side to the patch selected on the right side, or the working copy if it has been selected on the right.

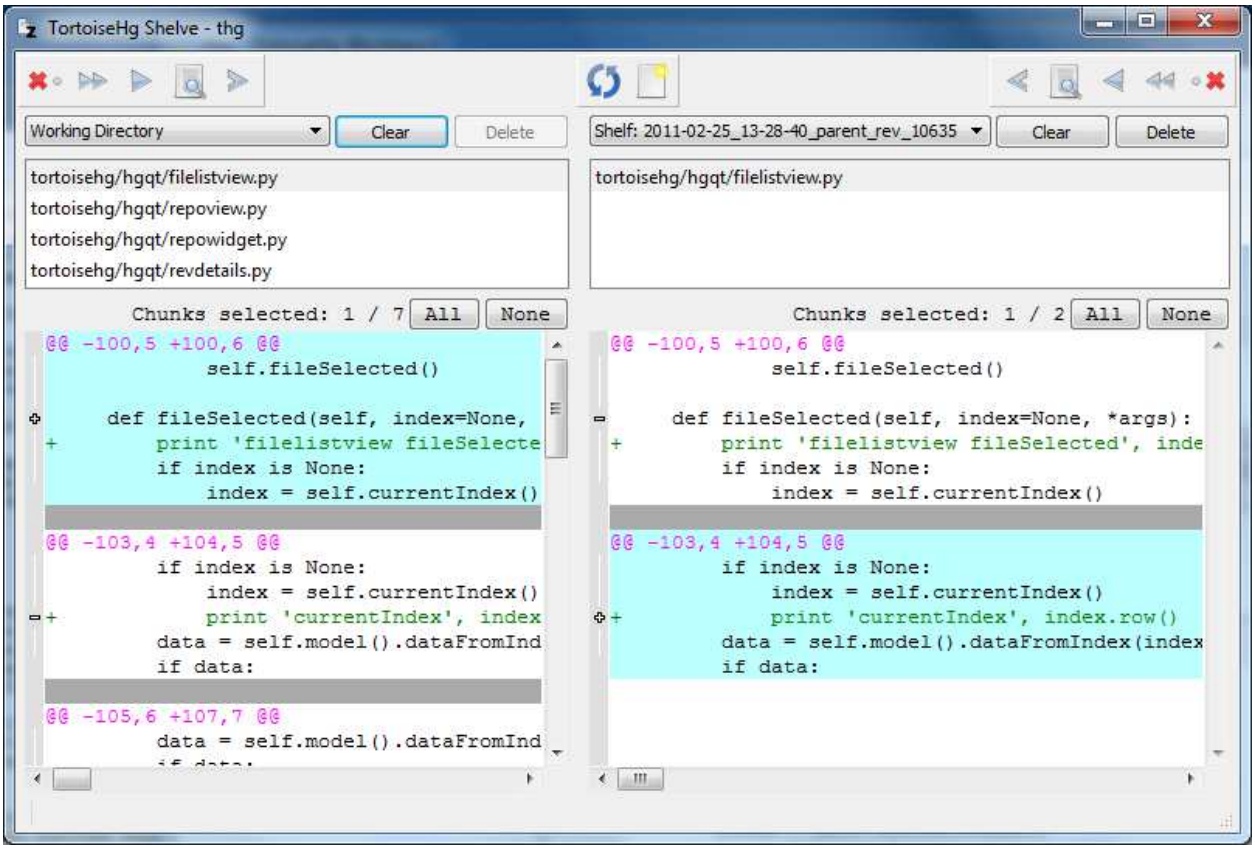


Figure 5.20: Shelve dialog

5.8.2 Patch Panes

The right and the left patch panes are identical save for the working copy changes are only available on the left. Selectable chunks are only displayed if the file is text and is in a modified state. Added or removed files can be shelved but parts of the file cannot be individually selected.

The **Clear** button will empty the currently selected patch or revert the entire working copy. The **Delete** button will delete the currently selected shelf patch.

Note: The **Delete** button is not sensitive when an MQ patch is selected. MQ patches must be deleted via qdelete using the Workbench context menu or the Patch Queue widget.

When right clicking on a file in the file list, you will get a context menu of commands.

Visual Diff Open the selected file in your default visual diff tool. Only enabled for working copy files.

Edit Local Open the working copy version of the selected file.

Revert to Revision Revert all changes to the selected file. Only enabled for working copy files.

5.8.3 Trashcan

The shelve tool is very conservative with your source and patch files. Before it modifies any file it makes a backup under `.hg/Trashcan`. This trashcan can be emptied by running the purge dialog from the Workbench **Repository –> Purge** menu option.

5.8.4 From command line

The shelve tool can be started from command line:

```
thg shelve

aliases: unshelve

shelve tool

use "thg -v help shelve" to show global options
```

5.9 Synchronize

The synchronize tool is used to transmit changesets between repositories or to email recipients.

Incoming show changesets that would be pulled from target repository, the changes in the target repository that are not in local repository

Pull pull incoming changesets from target repository

Outgoing show changesets that would be pushed to target repository, the changes in the local repository that are not in target repository

Push push outgoing changesets to target repository, make the local *tip* the new *tip* in the target repository

Email send outgoing changesets (to target repository) as email

Stop stop current operation

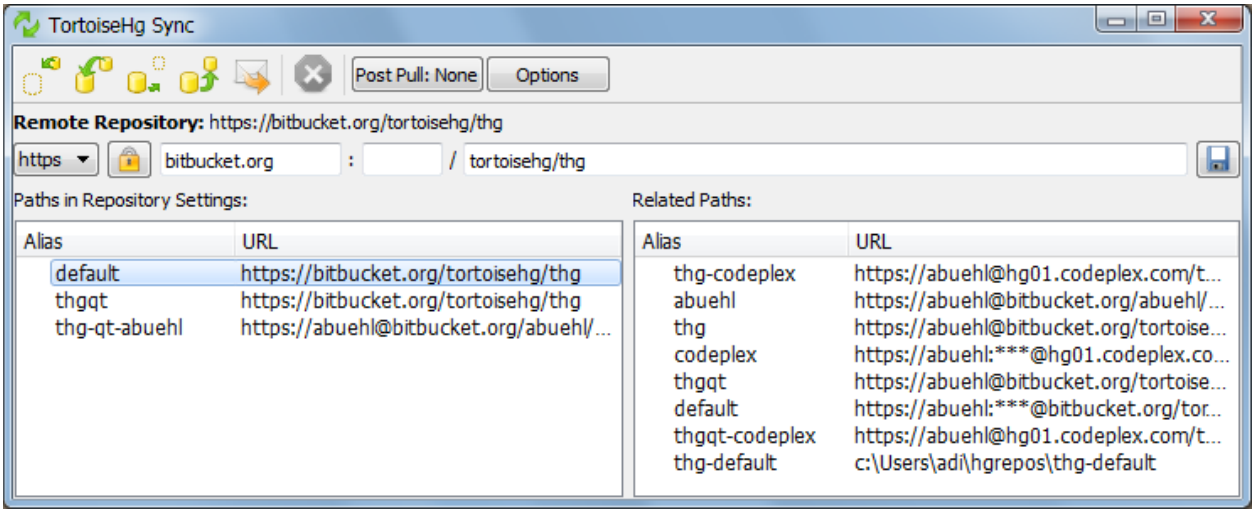


Figure 5.21: Synchronize dialog

The **Post Pull** dialog contains radio buttons for selecting the operation which is performed after a pull. If you open the configuration tool, you can select a default behavior for your user account and override that selection on a per-repository basis.

None No operations are performed after a pull. You will be allowed to view the pulled changesets in the log viewer, and you will have the option to update to the new tip if applicable.

Update Automatically update to the current branch tip if, and only if, new revisions were pulled into the local repository. This could trigger a merge if the pulled changes conflict with local uncommitted changes.

Fetch Equivalent to hg fetch. See the fetch extension documentation for its behavior. This feature is only available if the fetch extension has been enabled by the user.

Rebase Equivalent to pull –rebase. See the rebase extension documentation for its behavior. This feature is only available if the rebase extension has been enabled by the user.

Automatically resolve merge conflicts where possible If update or rebase are selected, a pull operation may result in a merge. If checked, Mercurial will try to resolve trivial merge conflicts without user interaction. If not checked, all merges will be interactive.

The **Options** dialog provides checkboxes for selecting infrequently used command options.

Allow push of a new branch allow a new named branch to be pushed

Force pull or push override warnings about multiple heads or unrelated repositories

Recurse into subdirectories incoming or outgoing commands can recurse into subdirectories and provide a full report

Temporarily disable configured proxy only sensitive when a web proxy is configured for the given repository. While checked it will disable that proxy.

Remote Command provides a –remotecmd argument

When the sync tool is opened within the Workbench, the toolbar has a **Target** checkbox. While checked, the target dropdown box is sensitive and the selected target revision, bookmark, or branch will be added to every synchronization command. When the sync tool is opened outside of the Workbench, the target checkbox and dropdown box is hidden. Clicking on a revision in the graph will update the values in the dropdown box. Holding `Alt` while clicking on a revision will select the revision without switching away from the sync tool tab.

Below the toolbar is the currently selected URL. All synchronization commands will use this URL. The general effect of the toolbar is that it can be read as a Mercurial command line. The tool buttons select the command, the **Post Pull** and **Options** dialog specify options, the target dropdown box can specify revisions, and finally the URL completes the command.

5.9.1 Adding an URL

By far the easiest way to add a new URL to your respository is to drag and drop the URL from another application, then press the save button and provide the URL an alias.

The two list panes display URLs that are stored in the current repository's configuration file (**Stored Paths**) and URLs that are stored in other related repositories that are listed in the Workbench repository registry (**Related Paths**). When the sync tool is opened outside of the Workbench, the **Related Paths** list will be empty.

Note: Being related means two repositories share at least a common root changeset. Cloned are obviously related. Push and pull operations require that repositories to be related, or that you use –force to override the relationship check.

The URL lists have a context menu that allows you to browse, open a terminal, or delete an URL from your local configuration file. The platform standard delete key sequence will also remove an URL.

5.9.2 Security

Mercurial (and TortoiseHg) support two secure protocols for exchanging data with remove servers. HTTPS (SSL) and SSH.

HTTPS

There are two asymmetrical parts to a secure HTTPS connection. The first part of the secure connection is authenticating the identification of the server. The second is authenticating yourself (the client) to the server, either via a username and passphrase or a certificate key.

Host Authentication

Prior to version 1.7, Mercurial ignored this half of HTTPS connection security. In version 1.7 it began warning that the server's certificate was not being verified. Starting with Mercurial version 1.7.3 (TortoiseHG 1.1.7), the binary installers begain to include a CA certificate file so that HTTPS server certificates could be verified by the standard certificate authorities. We download our certificate authority file from http://curl.haxx.se/ca/cacert.pem.

Mercurial version 1.7.5 introduced the ability to validate an HTTPS server's certificate against a stored fingerprint. TortoiseHg 2.0's synchronize tool has an HTTPS security dialog that allows you to select between using a host fingerprint or using the CA certificates.

In theory, a host fingerprint is more secure than the CA certificates if you do not necessarily trust all of the signing authorities listed in the cacert.pem file. However you must be sure that the fingerprint you store is the correct fingerprint for the server to which you believe you are communicating.

TortoiseHg 2.0 also allows you to select an insecure connection for a given host. This disables validation of the host's certificate but still uses an encrypted data stream (which was essentially the behavior of Mercurial pre-1.7 except for the warning messages).

User Authentication

There are several mechanisms available for authenticating yourself to an HTTPS server. The simplest is to allow Mercurial to prompt you for the username and passphrase. However this quickly grows old as the two prompts are always made separately and each push operation can require multiple connections to be established.

The next option is to encode the username in the URL so that Mercurial only prompts for a passphrase. This cuts the number of prompts in half, but is still annoying. If you do not wish to be prompted for the passphrase, it must be stored somewhere. Your choices, in increasing security, are:

1. encode the clear-text passphrase in each HTTPS URL in your repository configuration files
2. store the clear-text passphrase in your user configuration file
3. use the mercurial_keyring extension to store the passphrase cryptographically

Until recently, TortoiseHg only supported the first option in the graphical interface even though the second and third options were supported internally. TortoiseHg 2.0, we only support the latter two options in the graphical interface, and we do not allow the user configure the first option anymore. By default we strip the username and password off of URLs when they are saved.

To migrate from the first option to the later options, select an HTTPS URL in the sync tool, open the security dialog and enter a username and passphrase for the host if none are configured, and save. Next save the URL itself and allow the save dialog to strip the user authentication data from the URL.

Note: If the mercurial_keyring extension is enabled, the security dialog will not allow you to enter a passphrase since you do not want to store the passphrase in clear text in your configuration file if you are going to later store it cryptographically.

Options 2 and 3 use the [auth] section of your user configuration file to configure a single username and passphrase (or certificate key files) to authenticate to a given HTTPS hostname. The [auth] section supports many more configurations than this, see the man page for details.

Once the mercurial_keyring extension has been enabled (and all applications are restarted), you can remove the HTTPS passphrases from all of your configuration files. Mercurial will prompt for the passphrase once, then store it cryptographically using the best back-end it can find for your platform.

The mercurial_keyring extension requires the [auth] section to be configured for the host to which you are connecting, to provide the username. If your URL has an encoded username or passphrase, the [auth] section is ignored.

SSH

SSH is a symmetrical peer-to-peer secure tunnel. SSH clients and servers have their own key management systems, so Mercurial does not get involved with password prompts when SSH is used. This is problematic on Windows and thus TortoiseHg bundles the TortosePlink SSH client with its Windows installers. TortoisePlink is a port of the Plink SSH client that uses dialog prompts for host-key authorizations and passphrase prompts. TortoisePlink (developed by the TortoiseSVN project) can use the other SSH tools that are part of the Plink toolchain, including the Pageant key agent.

It is a known issue that TortoisePlink does not use compression in many scenarios, and thus is up to four times slower than openssh and other clients. TortoiseHg recommends the use of HTTPS for Windows clients.

See the FAQ for help if you have trouble connecting to ssh servers.

5.9.3 Email

The email dialog can be launched from two TortoiseHg tools.

1. The Workbench, in which case the user intends to email a selection of revisions.

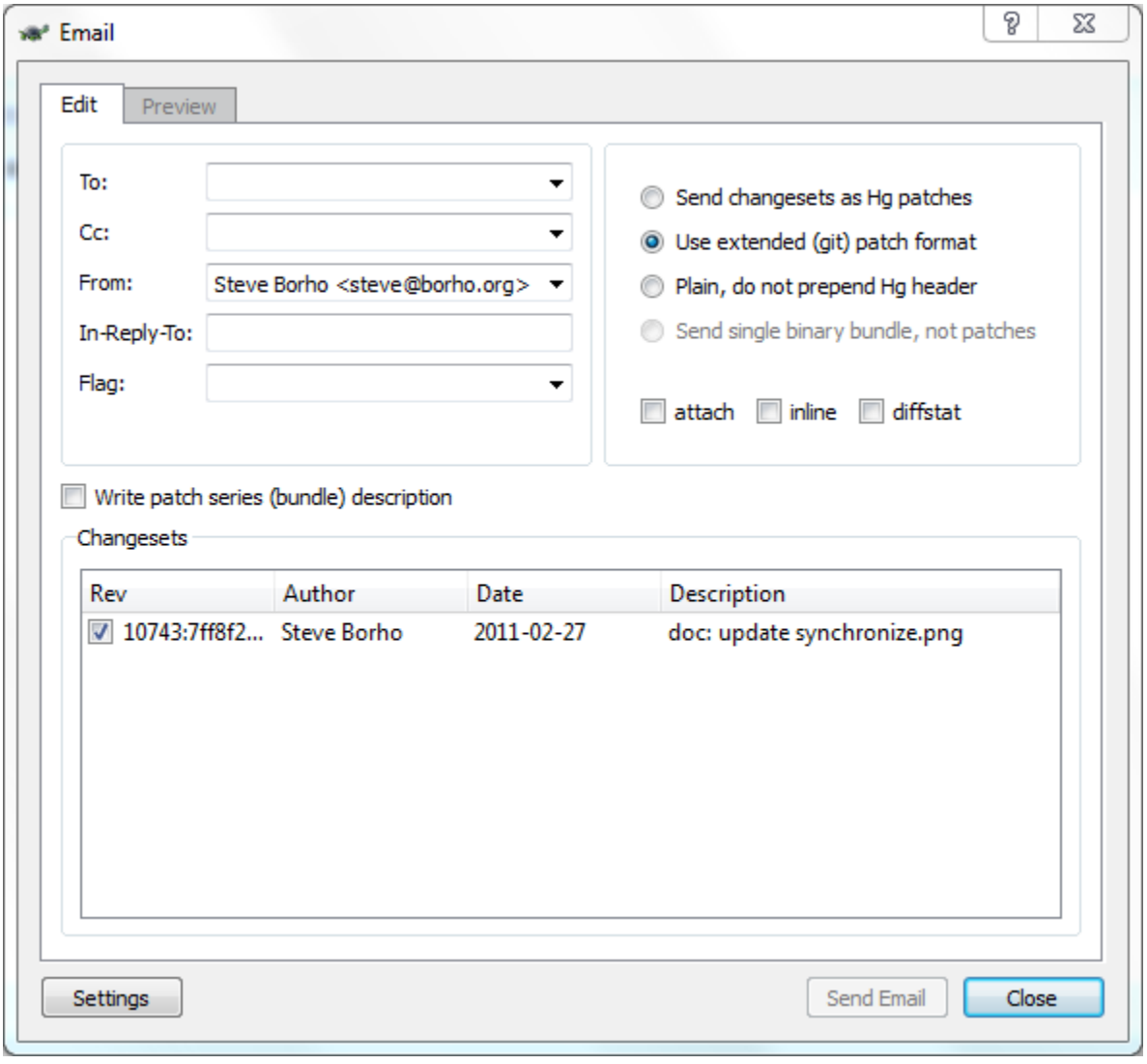


Figure 5.22: Email dialog

2. The synchronize tool, in which case the user intends to email all outgoing changes to the current target repository.

The **Send** button is obvious, and the **Configure** dialog predictably opens the TortoiseHg Settings dialog to the email tab where you can configure your SMTP settings and set default **To:** and **From:** addresses.

In-Reply-To: is used to make your patches properly threaded in mailing lists.

Please consult the Mercurial documentation for the differences between plain patches, Hg patches, Git patches, and bundles.

5.9.4 From command line

The synchronize tool can be started from command line

```
thg sync

aliases: synchronize

Synchronize with other repositories

use "thg -v help sync" to show global options
```

The syntax is simple, no options or parameters are needed, except the global options.

5.10 Serve

The serve tool is a wrapper for Mercurial's built-in web server. Once launched, a computer can connect to the http port and browse your repositories, perform clone, pull, or even push operations if enabled.

Toolbar buttons:

Start start the web server

Stop stop the web server

Configure Configure repository web style, description, and access policies

When the settings dialog is launched via the **Configure** button, it is run in the context of the current repository. Please visit the Mercurial wiki for detailed descriptions of the various web configurations.

In TortoiseHg 2.0, the serve tool natively supports collections of local repositories. Just drag them onto the web serve dialog while it is not running, or add them by hand using the editing buttons. The repository collections can be saved and reloaded.

5.10.1 From command line

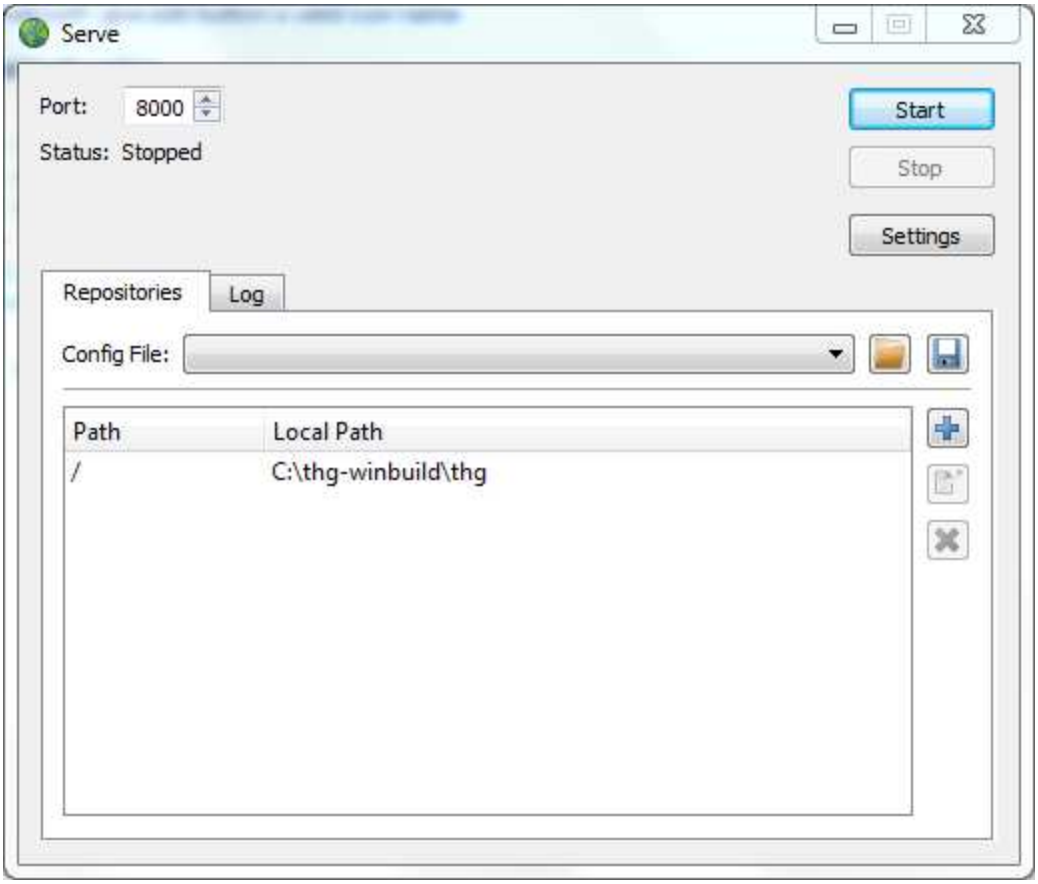
The server tool can be started from command line

```
thg serve [OPTION]...

start stand-alone webserver

options:

        --webdir-conf  name of the webdir config file
```


Serve
Port:
8000
Status: Stopped
Start
Stop
Settings
Repositories
Log
Config File:
Path
Local Path
/
C:\thg-winbuild\thg

```
use "thg -v help serve" to show global options
```

5.11 Detect Renames

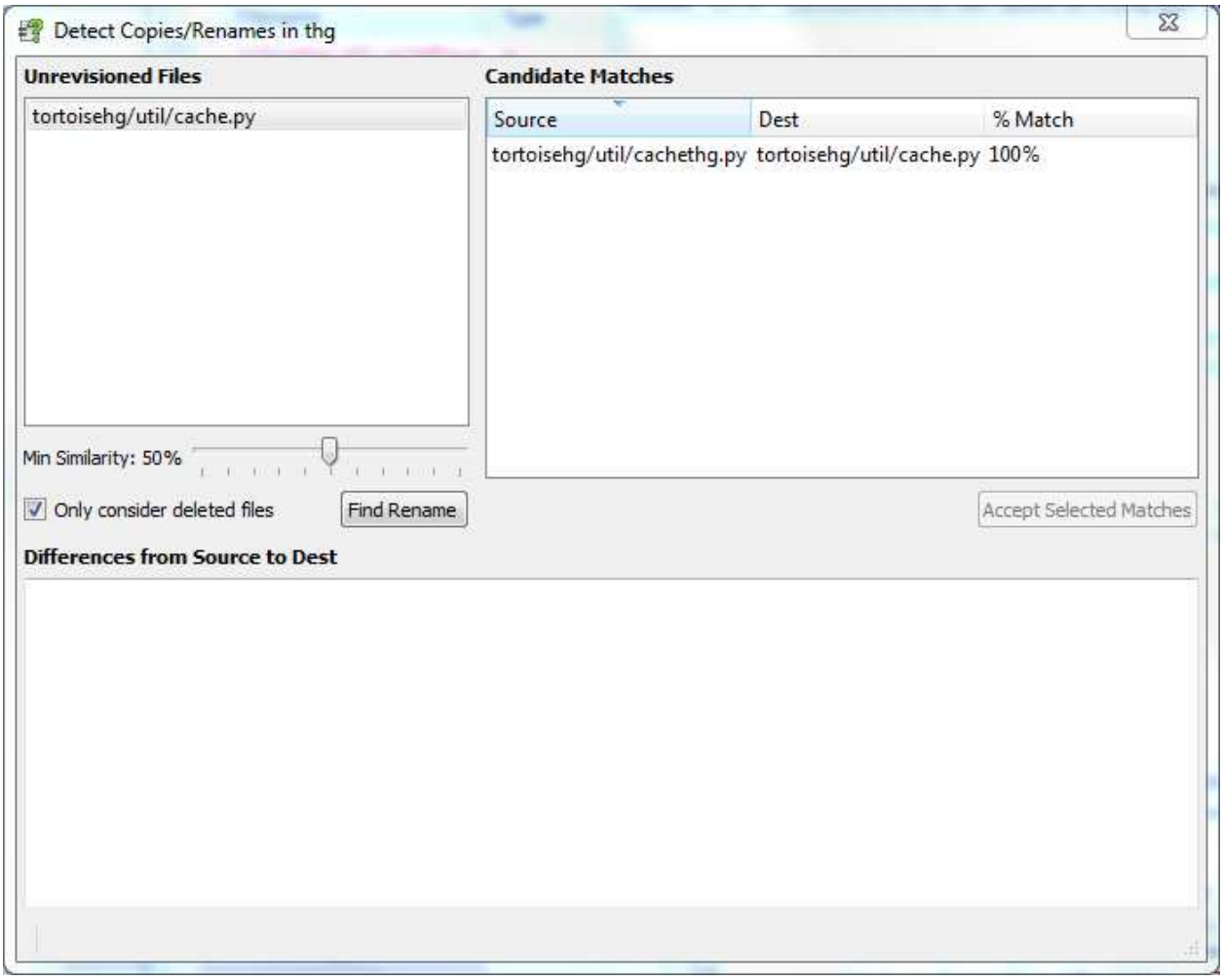


Figure 5.23: Rename Guessing Dialog

This dialog is used to find renames, moves, and/or copies that were done without Mercurial's knowledge. The dialog can be launched from the shell context menu, or from the status or commit tools via the context menu of an unknown file.

Follow these steps:

1. select one or more of the **Unrevisioned Files**
2. slide the **Min Similarity** bar to the percentage match you desire
3. uncheck **Only consider deleted files** to search for copies
4. press **Find Rename**
5. preview **Candidate Matches** and accept good matches

6. repeat until all unrevisioned files are matched

5.11.1 Candidate Matches

When you select a match in this list, the differences between the two files are shown in the bottom pane. Pressing **Accept Match** will record the rename or copy event with Mercurial.

5.11.2 From command line

The guess tool can be started from command line:

```
thg guess

guess previous renames or copies

use "thg -v help guess" to show global options
```

5.12 Ignore Filter

The ignore dialog is used to maintain your Mercurial repository's ignore filter, which can be found in an `.hgignore` file in the repository root. The dialog can be launched from the shell context menu, or from the status or commit tools via the context menu of an unknown file. The **Glob** combo allows to switch between glob or regexp patterns.

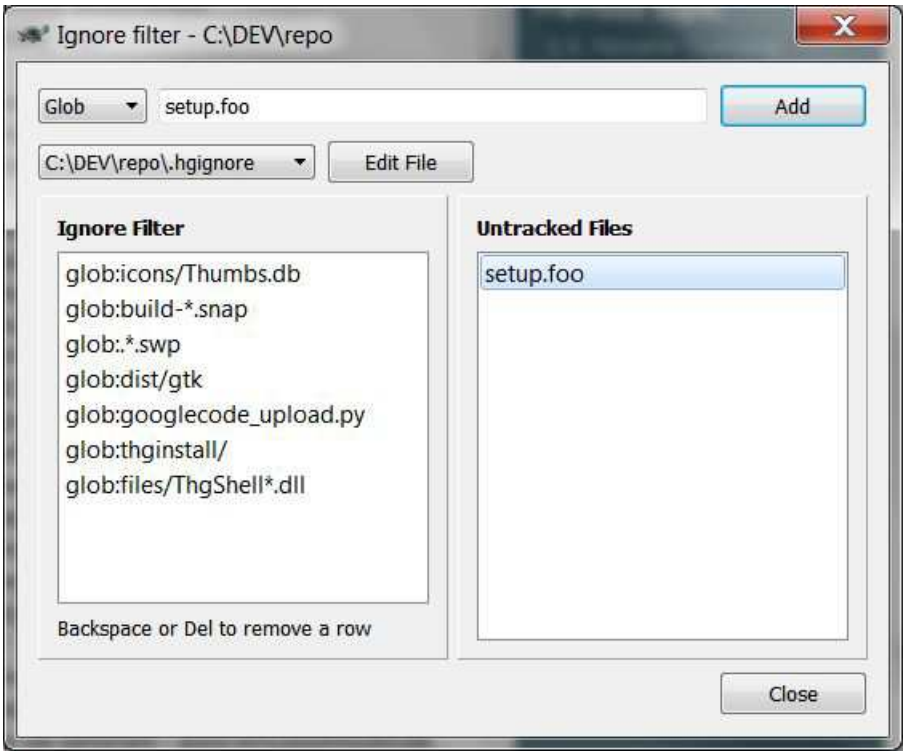


Figure 5.24: Ignore Filter Dialog

5.12.1 From command line

The ignore tool can be started from command line:

```
thg hgignore [FILE]

aliases: ignore, filter

ignore filter editor

use "thg -v help hgignore" to show global options
```

5.13 Archiving

You can choose from quite a lot of formats to make an archive of a repository. There are the usual compressed formats. But it's also possible to make a plain folder of files in another place than the Working Directory (it's like a clone, but without the .hg folder). Can be useful for doing builds for example (think 'svn export'). One can make an archive with only the files of the selected rev. Some people seem to use this to send changes to external people that do not have access to the repository.

From the changelog context menu in the Workbench select *Archive...* for the choosen changeset, or, within the folder, type **thg archive**.

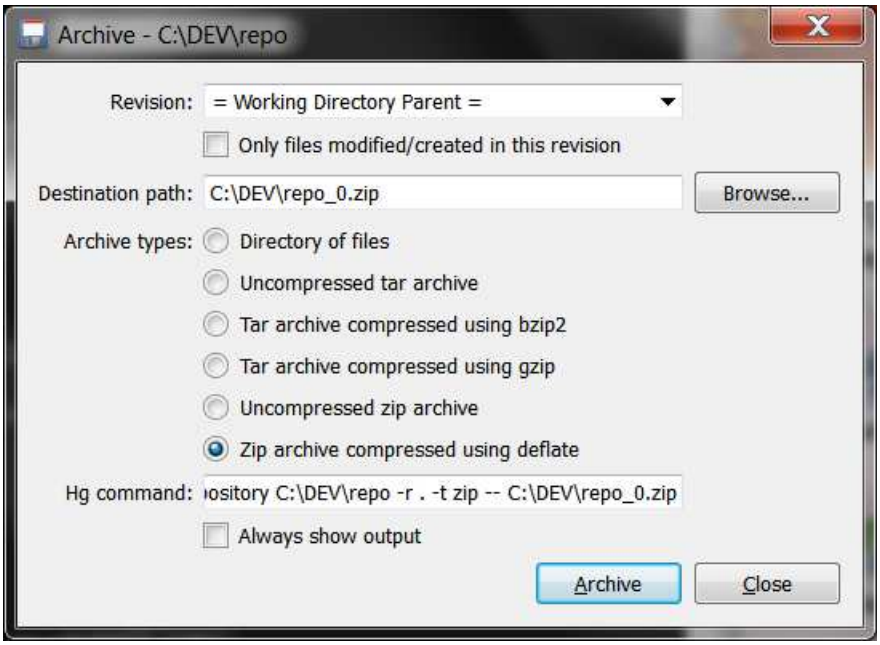


Figure 5.25: Repository Archive Dialog

Revision To select the revision you wish to archive or export.

Only files modified/created in this revision To limit the number of files in the archive.

Destination path The filename or directory where the archive will be created. It is filled with the name of the current repository, suffixed with the revision number of the selected revision, and has the appropriate extension of the selected archive type.

Archive types Here you can choose the type of archive to create, ranging from a plain folder with files to a variety of standard archive type.

Hg command This field displays the command that will be executed by the dialog.

Always show output To have an logging output pane with the results of the command while it runs.

Archiving a repository means create an archive file or subdirectory with the contents of the selected revision.

5.13.1 From command line

The archive tool can be started from command line

```
thg archive
```

The syntax is

```
thg archive -r [REV] -t [TYPE] [DEST]
```

where [REV] is the revision to archive, [TYPE] is the type of archive to create, and [DEST] is the name of the file or filder to create.

See hg.1.html#archive for details, or type **hg help archive** at the command line or in the Output Log of the Workbench.

CHAPTER

SIX

SETTINGS

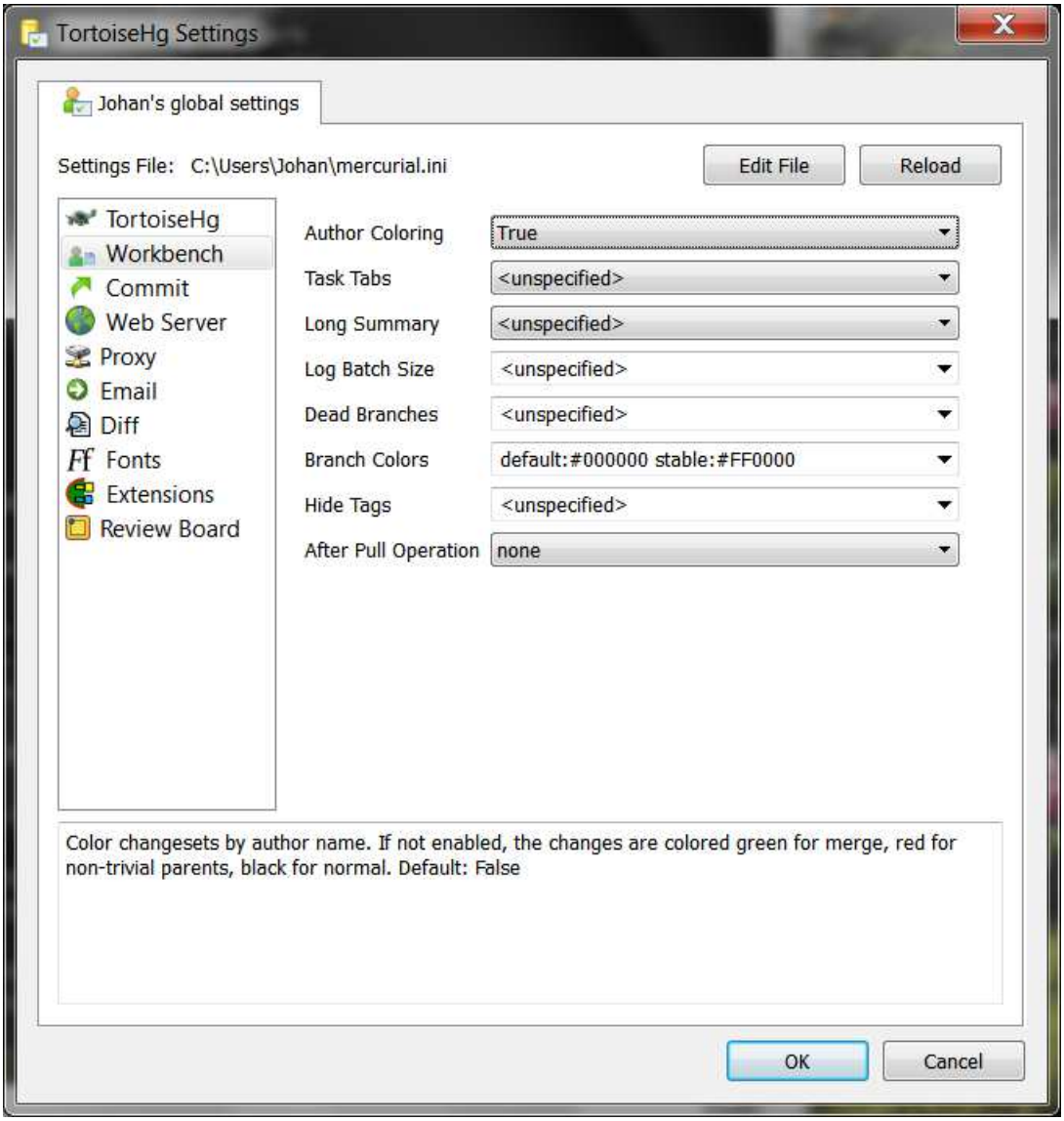


Figure 6.1: Settings dialog

The Settings dialog is used to configure both TortoiseHg and the underlying Mercurial DVCS. Since TortoiseHg uses

Mercurial's underlying configuration system to store and retrieve its settings, these are essentially the same thing.

Mercurial on Windows has a three-tier configuration system.

1. A site-wide configuration file in `C:\Program Files\TortoiseHg\Mercurial.ini` This file is read first and thus has the lowest priority.
2. A per-user configuration file in `C:\Documents and Settings\username\Mercurial.ini` This file is read second and thus can override settings in the site-wide configuration file.
3. A per-repository configuration file in `repo-root\.hg\hgrc` This file is read last and can override site-wide and user global settings.

The site-wide file can be overwritten on upgrades so it is recommended that you do not make changes to this file. Instead, you should make changes to your user `Mercurial.ini` and/or the repository `hgrc` file. The TortoiseHg Settings dialog enforces this suggestion by only operating in two modes:

Global edits your user `Mercurial.ini` file

Repository edits a repository `.hg/hgrc` file

You may toggle between the two modes using the combo box at the top of the dialog, or directly edit the file in your configured visual editor.

Most TortoiseHg users will want to store all configurables in their global user settings, and only use the repository hgrc to store paths (remote repository aliases) and web settings, though it is possible to override many configurables per-repository (a common example is to configure a username for use in a specific repository). Also note that the user and repository configuration files may not exist until you run the Settings dialog for the first time.

6.1 Tabs

The Settings tool is a tabbed application.

Each tab corresponds roughly to a section of your `Mercurial.ini` file, though there is a certain amount of overlap. Some sections were split across multiple tabs for clarity.

Every tab but **Sync** has the same format, a list of configurable options with a drop-down combo box with possible values and a history of options you have used for that setting. The configurable name (label) has a tooltip which describes in more detail what you are configuring and its default value. The description of the currently focused configurable is also shown in a text box at the bottom of the dialog.

Please consult the Mercurial wiki for more detailed information about these configurables (except for the first three tabs: **TortoiseHg**, **Commit**, **Changelog**, which are specific to TortoiseHg).

6.1.1 TortoiseHg

3-way Merge Tool: Graphical merge program for resolving merge conflicts. If left unspecified, Mercurial will use the first applicable tool it finds on your system or use its internal merge tool that leaves conflict markers in place. Chose **internal:merge** to force conflict markers, **internal:prompt** to always select local or other, or **internal:dump** to leave files in the working directory for manual merging.

Visual Diff Tool: Specify visual diff tool as described in the [merge-tools] section of your Mercurial configuration files. If left unspecified, TortoiseHg will use the selected merge tool. Failing that it uses the first applicable tool it finds.

Skip Diff Window: Bypass the builtin visual diff dialog and directly use your visual diff tool's directory diff feature. Only enable this feature if you know your diff tool has a valid extdiff configuration. Default: False.

Visual Editor: Specify the visual editor used to view files, etc.

CLI Editor: The editor to use during a commit and other instances where Mercurial needs multiline input from the user. Only used by command line interface commands.

Tab Width: Specify the number of spaces that tabs expand to in various TortoiseHg windows. Default: Not expanded.

Max Diff Size: The maximum size file (in KB) that TortoiseHg will show changes for in the changelog, status, and commit windows. A value of zero implies no limit. Default: 1024 (1MB).

Bottom Diffs: Show the diff panel below the file list in status, shelve, and commit dialogs. Default: False (show diffs to right of file list).

Capture stderr: Redirect stderr to a buffer which is parsed at the end of the process for runtime errors. Default: True.

Fork GUI: When running thg from the command line, fork a background process to run graphical dialogs. Default: True.

Full Path Title: Show a full directory path of the repository in the dialog title instead of just the root directory name. Default: False

Spell Check Language: Default language for spell check. System language is used if not specified. Examples: en, en_GB, en_US. Spell checking requires gtkspell, which is only available on Gnome PCs.

6.1.2 Commit

Username: Name associated with commits.

Summary Line Length: Maximum length of the commit message summary line. If set, TortoiseHg will issue a warning if the summary line is too long or not separated by a blank line. Default: 0 (unenforced).

Message Line Length: Word wrap length of the commit message. If set, the popup menu can be used to format the message and a warning will be issued if any lines are too long at commit. Default: 0 (unenforced).

Push After Commit: Attempt to push to default push target after every successful commit. Default: False

Auto Commit List: Comma separated list of files that are automatically included in every commit. Intended for use only as a repository setting. Default: None

Auto Exclude List: Comma separated list of files that are automatically unchecked when the status, commit, and shelve dialogs are opened. Default: None

6.1.3 Changelog

Author Coloring: Color changesets by author name. If not enabled, the changes are colored green for merge, red for non-trivial parents, black for normal. Default: False.

Long Summary: If true, concatenate multiple lines of changeset summary until they reach 80 characters. Default: False.

Log Batch Size: The number of revisions to read and display in the changelog viewer in a single batch. Default: 500.

Dead Branches: Comma separated list of branch names that should be ignored when building a list of branch names for a repository. Default: None

Branch Colors: Space separated list of branch names and colors of the form branch:#XXXXXX. Spaces and colons in the branch name must be escaped using a backslash (\). Likewise some other characters can be escaped in this way, e.g. \u0040 will be decoded to the @ character, and \n to a linefeed. Default: None

Hide Tags: Space separated list of tags that will not be shown. Useful example: Specify "qbase qparent qtip" to hide the standard tags inserted by the Mercurial Queues Extension. Default: None.

6.1.4 Web

Name: Repository name to use in the web interface. Default is the working directory.

Description: Textual description of the repository's purpose or contents.

Contact: Name or email address of the person in charge of the repository.

Style: Which template map style to use.

Archive Formats: Comma separated list of archive formats allowed for downloading.

Port: Port to listen on.

Push Requires SSL: Whether to require that inbound pushes be transported over SSL to prevent password sniffing.

Stripes: How many lines a "zebra stripe" should span in multiline output. Default is 1; set to 0 to disable.

Max Files: Maximum number of files to list per changeset.

Max Changes: Maximum number of changes to list on the changelog.

Allow Push: Whether to allow pushing to the repository. If empty or not set, push is not allowed. If the special value "*", any remote user can push, including unauthenticated users. Otherwise, the remote user must have been authenticated, and the authenticated user name must be present in this list (separated by whitespace or ","). The contents of the allow_push list are examined after the deny_push list.

Deny Push: Whether to deny pushing to the repository. If empty or not set, push is not denied. If the special value "*", all remote users are denied push. Otherwise, unauthenticated users are all denied, and any authenticated user name present in this list (separated by whitespace or ",") is also denied. The contents of the deny_push list are examined before the allow_push list.

Encoding: Character encoding name.

6.1.5 Proxy

Host: Host name and (optional) port of proxy server, for example `myproxy:8000`.

Bypass List: Optional. Comma-separated list of host names that should bypass the proxy.

User: Optional. User name to authenticate with at the proxy server.

Password: Optional. Password to authenticate with at the proxy server.

6.1.6 Email

From: Email address to use in the "From" header and for the SMTP envelope.

To: Comma-separated list of recipient email addresses.

Cc: Comma-separated list of carbon copy recipient email addresses.

Bcc: Comma-separated list of blind carbon copy recipient email addresses.

method: Optional. Method to use to send email messages. If value is "smtp" (default), use SMTP (configured below). Otherwise, use as name of program to run that acts like sendmail (takes **-f** option for sender, list of recipients on command line, message on stdin). Normally, setting this to `sendmail` or `/usr/sbin/sendmail` is enough to use sendmail to send messages.

SMTP Host: Host name of mail server.

SMTP Port: Port to connect to on mail server. Default: 25.

SMTP TLS: Connect to mail server using TLS. Default: False.

SMTP Username: Username to authenticate to mail server with.

SMTP Password: Password to authenticate to mail server with.

Local Hostname: Hostname the sender can use to identify itself to the mail server.

6.1.7 Diff

Patch EOL: Normalize file line endings during and after patch to lf or crlf. Strict does no normalization. Auto (introduced in hg 1.5) does per-file detection and is the recommended setting. Default: strict

Git Format: Use git extended diff header format. Default: False.

No Dates: Do not include modification dates in diff headers. Default: False.

Show Function: Show which function each change is in. Default: False.

Ignore White Space: Ignore white space when comparing lines. Default: False.

Ignore WS Amount: Ignore changes in the amount of white space. Default: False.

Ignore Blank Lines: Ignore changes whose lines are all blank. Default: False.

Coloring Style: Adjust the coloring style of diff lines in the changeset viewer. Default: foreground.

6.1.8 Font

Theme default fonts Use default fonts based on current GTK theme.

Preset fonts: Select preset fonts from drop-down combo. These font sets are tuned specifically for each language and/or environment.

Custom fonts: Set font names and sizes individually for each usage place.

The group which contains drop-down combo entries under **Custom fonts:** radio button is enabled when you activate it.

Commit Message: Font used in changeset viewer and commit log text. Default: monospace 10.

Diff Text: Font used for diffs in status and commit tools. Default: monospace 10.

File List: Font used in file lists in status and commit tools. Default: sans 9.

Command Output: Font used in command output window. Default: monospace 10.

6.2 Keyboard navigation

`Ctrl-Enter` Apply changes and close the tool, the equivalent of pressing the 'Ok' button.

6.3 From command line

The setting dialog can be started from command line

```
thg repoconfig
```

for the repository settings (`.hg/hgrc` file) or

```
thg userconfig
```

for the user configuration (`Mercurial.ini` file).

The syntax is simple, no options or parameters are needed, except the global options.

CHAPTER

SEVEN

PATCHES

7.1 Defining a patch

These links are recommended reading for understanding the history and nature of patches and how they can be used with Mercurial.

- The patch management problem
- Understanding patches
- More about patches

7.2 Pitfalls

The standard patch format cannot describe binary files, renames, copies, or permission changes. If your patch needs to record any of those things, you will need to enable **git** patches via:

```
[diff]
git=True
```

Mercurial 1.5 improves its behavior in this regard. It will warn you when git diffs are required, or sometimes upgrade to the git format automatically. See also the diff section of the hgrc documentation.

Mercurial's patch routines do not deal well with mixed EOLN between source files and patches. The **patch.eol** setting was introduced in 1.3 to improve this situation:

```
[patch]
eol = auto #strict, lf, or crlf
```

Note: When eol is set to *auto*, the patching engine will preserve the line endings of the patched file regardless of the line endings in the patch itself. You almost always want eol to be configured to *auto*. The only downside is that you cannot make a patch that changes the line endings of a source file.

See also the patch section of the hgrc documentation.

Applying a patch is not a foolproof operation. If the source file has diverged from the file that was used to create the patch, there may be conflicts during the patch application. These are written to a file with an .rej extension. TortoiseHg 2.0 includes a **thg rejects** command that can aid in the merging of the rejected chunks into the source file.

7.3 Export Patches

7.3.1 Changeset

To export a changeset as a patch file, use the changeset context menu of the Workbench to select *Export* → *Export Patch*. It writes the changeset to a file in the repository root folder.

7.3.2 Changeset Ranges

Select the start and end of a range of changesets in the Workbench and open the special revision range context menu. From this menu you can generate patches, generate a bundle, send emails, or visually diff the accumulated changes.

This is a very powerful feature and there is no restriction on the base and target changesets you can select.

7.3.3 Email

To send a changeset as an email, use the changeset context menu of the Workbench. *Export* → *Email Patch*. This opens the e-mail dialog for this single changeset.

To send a changeset range, use the changeset range selection feature of the Workbench and select *Email selected...* or *Email DAG range....*

Lastly, you can use the **Email** button on the sync tab of the Workbench to email all outgoing changes to the selected remote repository.

Note: You must configure SMTP to send patches via email

7.4 Import Patches

The import dialog can be opened from the **Repository** menu of the Workbench, or via **thg import**. The dialog supports file and directory drag and drop.

You have the choice of importing directly into the repository, the working folder, a shelf file, or your patch queue.

Note: Importing a patch requires a clean working directory state. You must commit, revert, or shelve changes before importing a patch.

Note: If a patch does not import itself cleanly into the repository, the recommended recourse is to import the patch into your patch queue (qimport) and then qpush the patch. This uses TortoiseHg patch rejection dialog and preserves the meta-data in the patch header. Do not forget to qrefresh after resolving the rejected chunks.

Warning: If the patch you are importing does not have a commit message, Mercurial will try to launch your editor, just as if you had tried to import the patch from the command line. Your ui.editor needs to be a GUI app to make this work correctly.

7.5 Patch Queues

When MQ is enabled, several Workbench features are exposed. Context menu options are exposed in the changeset menus, your patch queue is graphed together with your repository's history, and a Patch Queue task tab is activated.

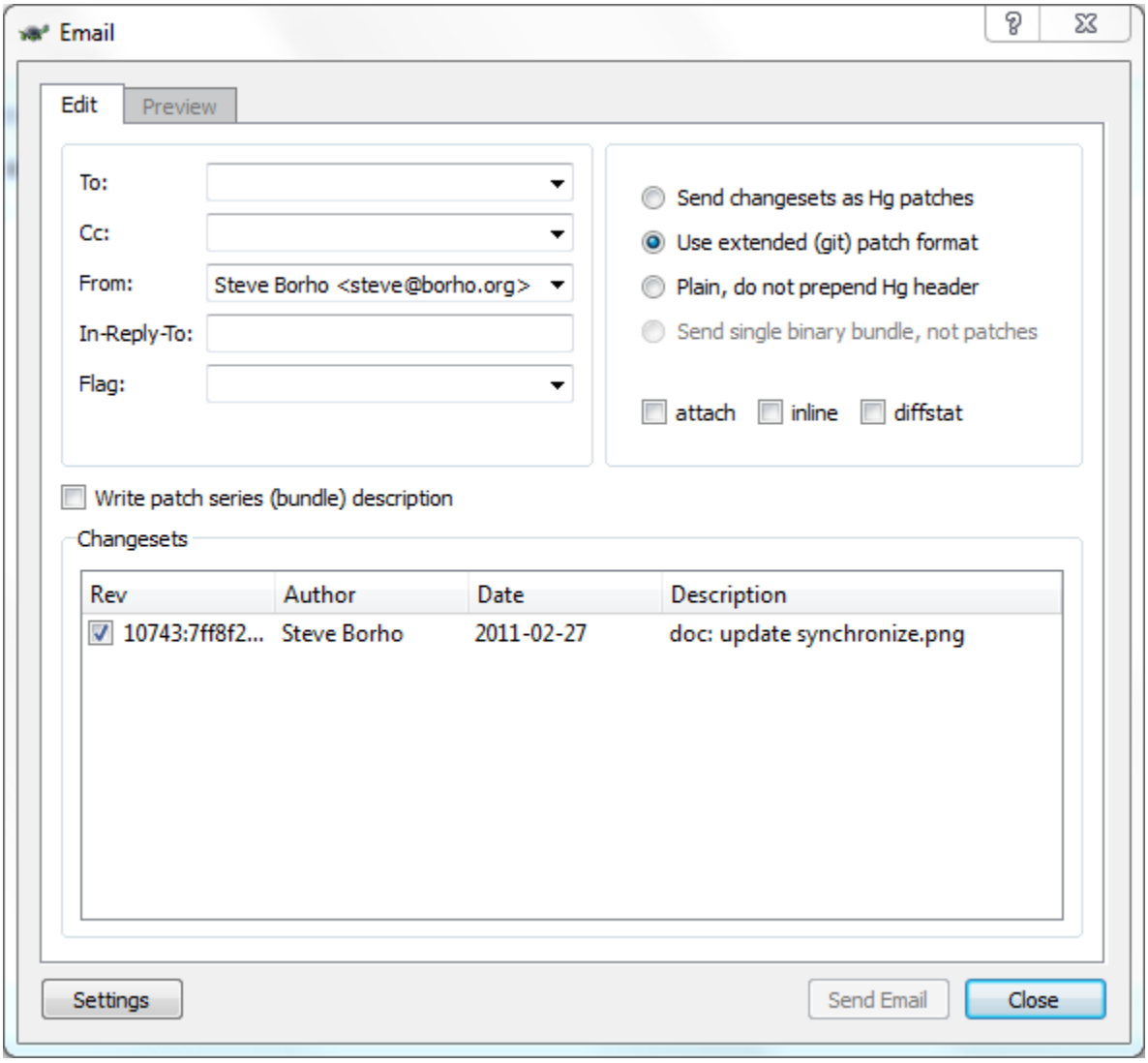


Figure 7.1: Email dialog of Workbench

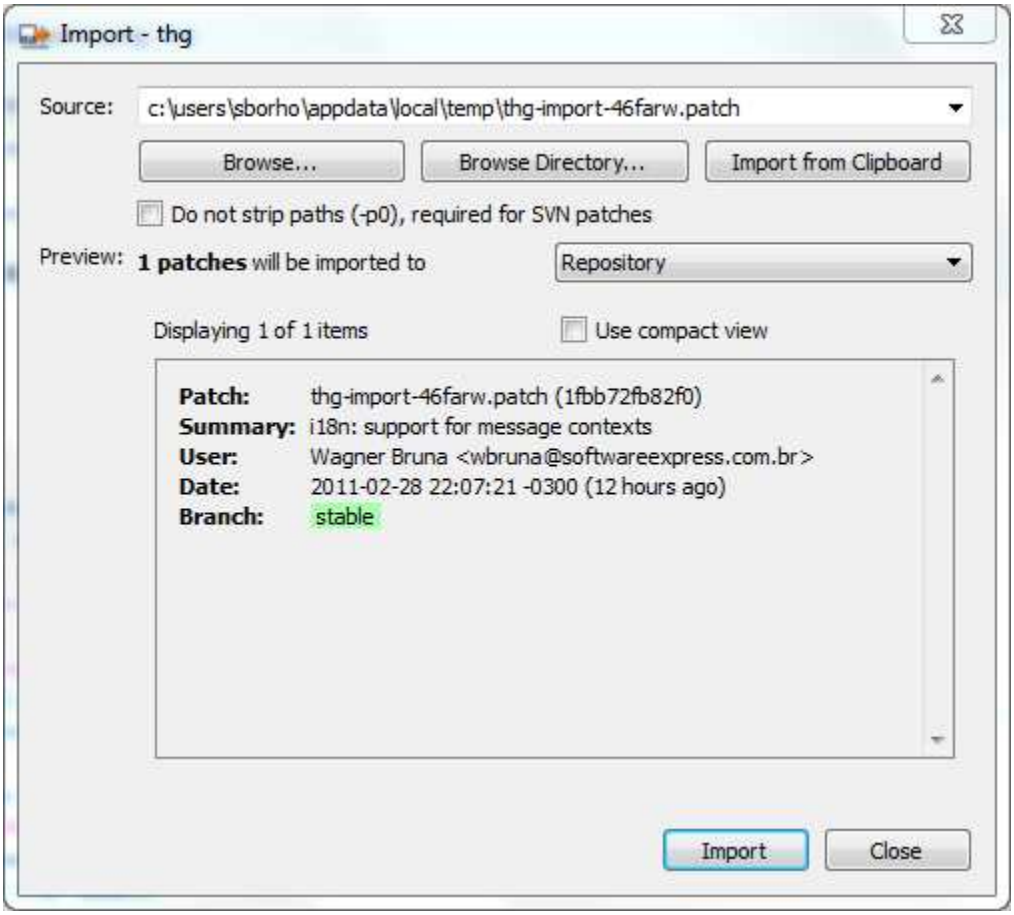


Figure 7.2: Import dialog of the WorkBench

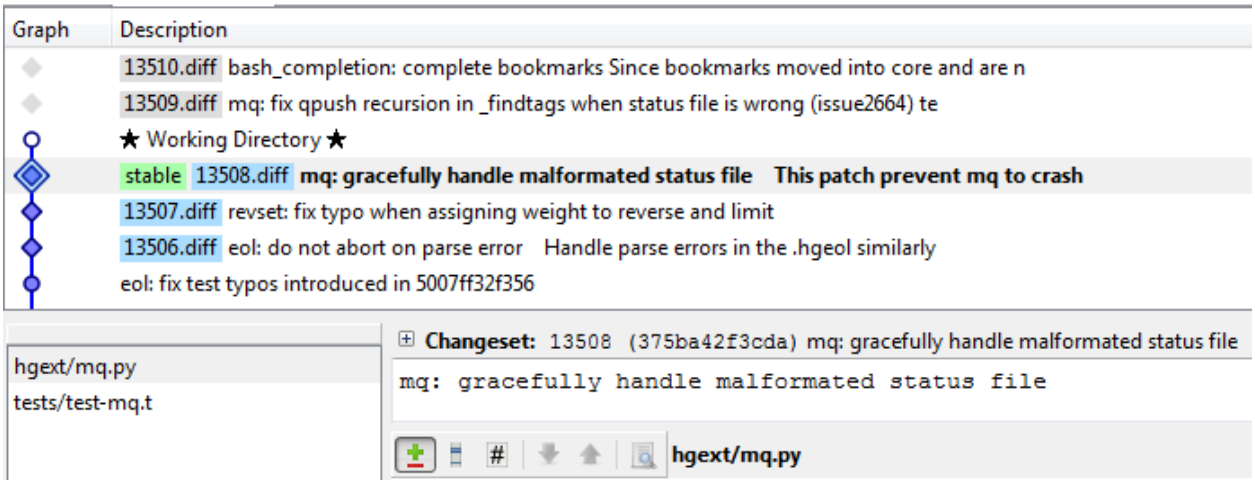


Figure 7.3: A patch queue in the repository graph

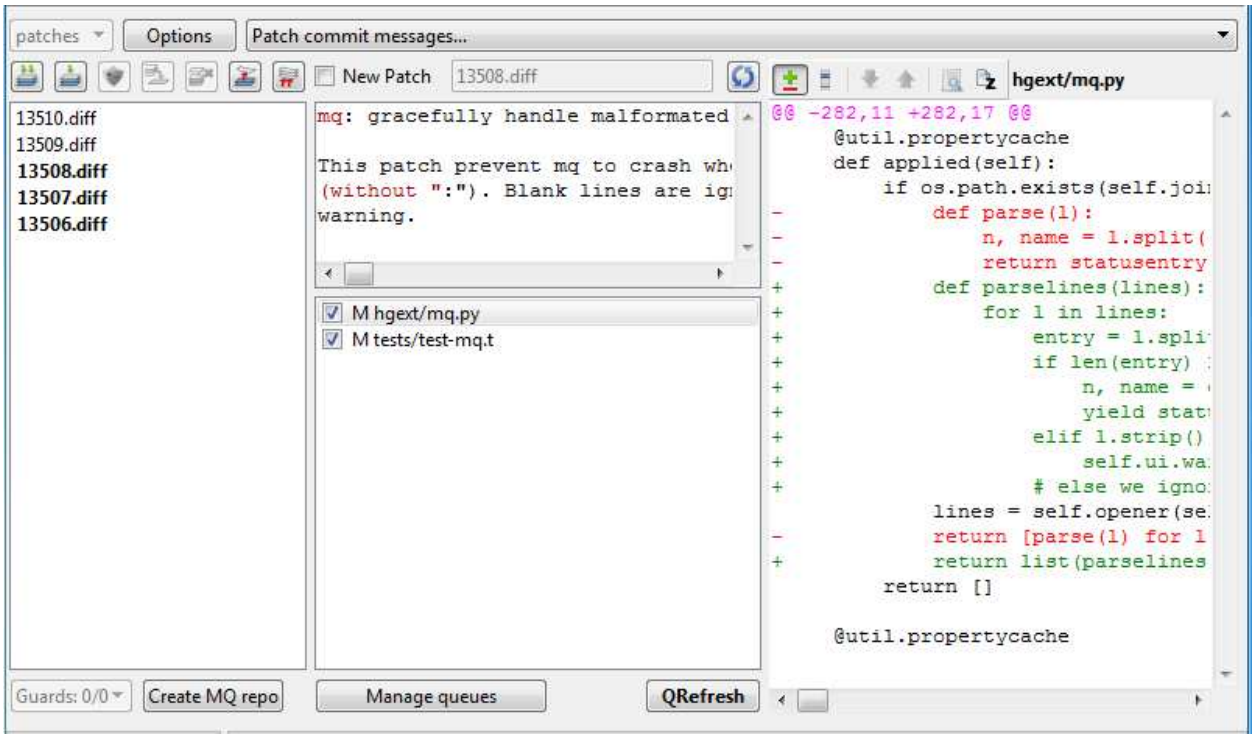


Figure 7.4: MQ tool or task tab

The **Patch Queue** task tab is also available as **thg mq** from the command line. Most MQ features are available from the Patch Queue task tab, though a few features (qimport -r/qfinish) are only supported via revision graph context menus. Much functionality is supported in both places.

Double clicking on an unapplied patch, an applied patch other than the current qtip, or the qparent triggers a qgoto command; making the double clicked revision the current patch. Double clicking on any other revision will trigger a visual diff of that revision.

In previous releases, the commit tool was a central feature in MQ use. In TortoiseHg 2.0, the commit tool is almost completely unaware of MQ. It only knows to not allow commits if the current working parent is an applied patch. Instead, MQ functionality has been focused into the Workbench, Patch Queue task tab, and the shelve tool.

Note: The Workbench must be restarted after enabling or disabling the MQ extension in a repository. This is true of most extensions.

Note: It is recommended to learn the MQ extension before using the MQ features of the Workbench.

7.6 Patch Rejects

As explained previously, patches are not guaranteed to apply cleanly to their intended source files. Prior to TortoiseHg 2.0, the only recourse available when patch chunks were rejected was to open the source file and the rejects file in an editor and manually fixup the rejected chunks.

TortoiseHg 2.0 introduces a dialog that makes this a little bit easier. If the shelve tool detects chunk rejections, it offers to open the rejected chunks in the rejects editor. The MQ tool also does this for qpush commands.

The rejects editor is very basic. Your source file is opened in a QScintilla2 window for edit. Below the source file is the list of chunks that failed to apply to this file. When you click on a chunk in the list the editor jumps to the line where the chunk context was supposed to match. It is up to you to figure out why the chunk did not apply and to resolve it

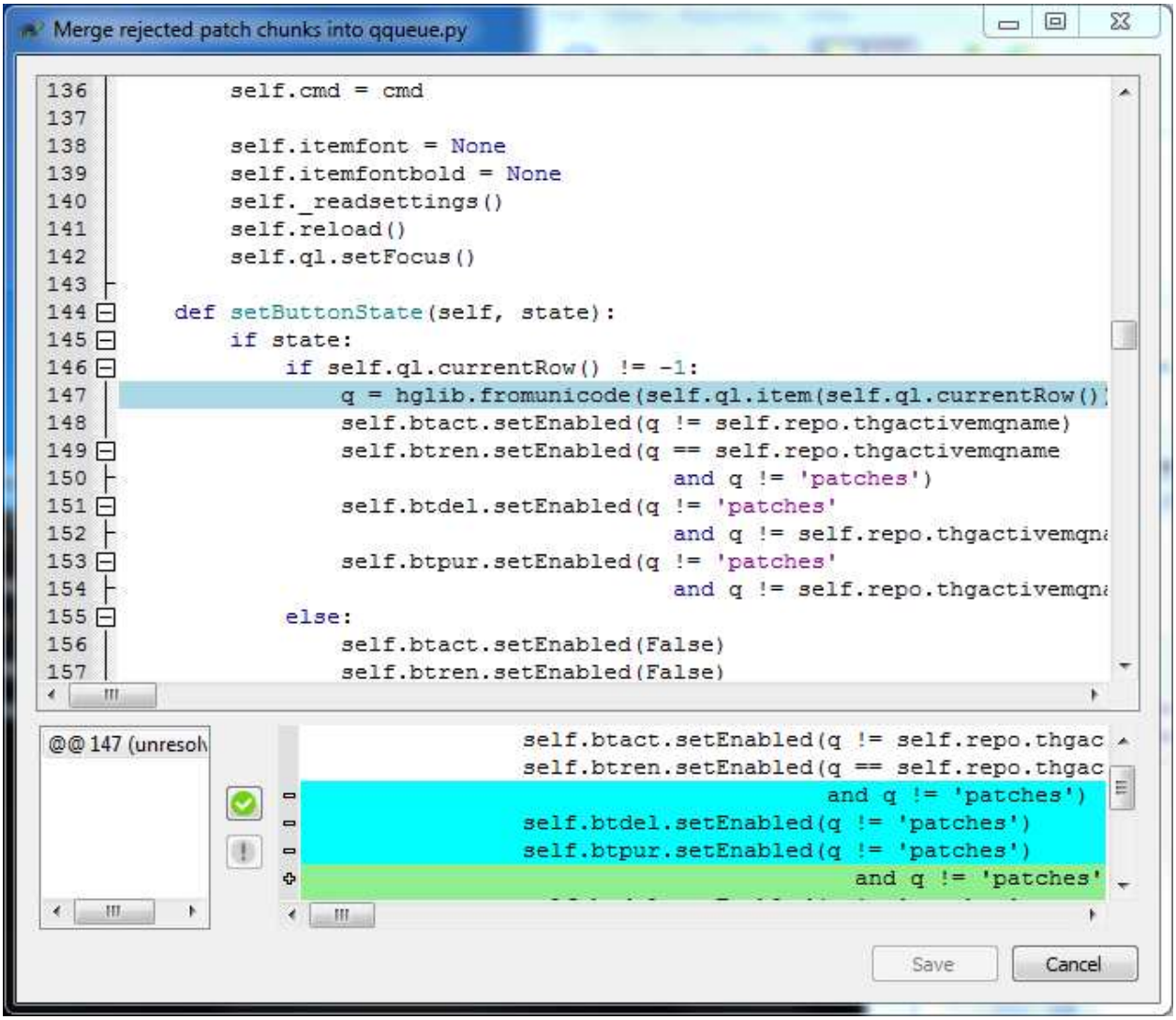


Figure 7.5: Resolve rejected patch chunks

(perhaps even by ignoring the chunk). The resolved/unresolved states are for your own book keeping, so you know when all of the chunks have been dealt with. Once you have marked all of the chunks resolved, the **Save** button will become sensitive.

CHAPTER

EIGHT

EXTENSIONS

This chapter describes Mercurial extensions that are shipped with TortoiseHg binary packages for Windows. These external extensions are included as a convenience to users, so they can be easily enabled as soon as they are needed.

8.1 Hgfold

hgfold is a Mercurial extension that helps Windows users deal with filename case collisions on VFAT and NTFS.

It adds options to the following Mercurial commands. Type **hg help <command>** for more information:

```
up    - allows you to update to a revision with filename collisions
merge - allows you to merge with a changeset that would create filename collisions
```

The extension does not currently do anything to prevent filename collisions. See discussion on the Mercurial Wiki

Installation

To test the use of this plugin, you can specify it on the Mercurial command line like this:

```
hg --config "extensions.fold=" status
```

You may want to add it to your Mercurial.ini or a repository's hgrc like this:

```
[extensions]
fold=
```

If you do this, you can omit the –config command-line option.

Warnings

Like all merge operations, fold.py has to change the parents of the working directory. It is still in early testing, so use with caution.

If you get an error about an unknown changeset after running **hg recover** try **hg debugsetparents <number of tip revision>**. You can find the number of the tip revision by running **hg log -l 2**.

8.2 Perfarce

Perfarce home page.

This extension is documented in *Perfarce (Perforce)* section of *Use with other VCS systems* chapter.

8.3 HGEOL

The hgeol extension is the eventual successor to the win32text extension. It tries to resolve the EOLN compatibility problems in a more complete and robust fashion. Instead of documenting it here, we will link to it's online documents which are continually evolving.

- EOLTranslationPlan
- Source code

8.4 Mercurial-Keyring

- Mercurial Keyring home page
- Keyring Extension wiki page

Keyring extension uses services of the keyring library to securely save authentication passwords (HTTP/HTTPS and SMTP) using system specific password database (Gnome Keyring, KDE KWallet, OSXKeyChain, dedicated solutions for Win32 and command line).

What it does

The extension prompts for the HTTP password on the first pull/push to/from given remote repository (just like it is done by default), but saves the password (keyed by the combination of username and remote repository url) in the password database. On the next run it checks for the username in `.hg/hgrc`, then for suitable password in the password database, and uses those credentials if found.

Similarly, while sending emails via SMTP server which requires authorization, it prompts for the password on first use of given server, then saves it in the password database and reuses on successive runs.

In case password turns out incorrect (either because it was invalid, or because it was changed on the server) it just prompts the user again.

Installation

First, the extension must be enabled in your Mercurial.ini file as:

```
[extensions]
mercurial_keyring=
```

Password backend configuration

The most appropriate password backend should usually be picked automatically, without configuration. Still, if necessary, it can be configured using `~/keyringrc.cfg` file (`keyringrc.cfg` in the home directory of the current user). Refer to keyring docs for more details.

Note: On Windows XP and above, your encrypted passwords are stored in the credentials subsystem using CredRead and CredWrite

Note: On Windows 2K, the encrypted passwords are stored in the system registry under HKCU\Software\Mercurial\Keyring.

Repository configuration (HTTP)

Edit repository-local `.hg/hgrc` and save there the remote repository path and the username, but do not save the password. For example:

```
[paths]
myremote = https://my.server.com/hgrepo/someproject

[auth]
myremote.schemes = http https
myremote.prefix = my.server.com/hgrepo
myremote.username = mekk
```

Simpler form with url-embedded name can also be used:

```
[paths]
bitbucket = https://User@bitbucket.org/User/project_name/
```

Note: If both username and password are given in `.hg/hgrc`, extension will use them without using the password database. If username is not given, extension will prompt for credentials every time, also without saving the password. So, in both cases, it is effectively reverting to the default behaviour.

Consult [auth] section documentation for more details.

Repository configuration (SMTP)

Edit either repository-local `.hg/hgrc`, or `~/.hgrc` (the latter is usually preferable) and set there all standard email and smtp properties, including smtp username, but without smtp password. For example:

```
[email]
method = smtp
from = Joe Doe <Joe.Doe@remote.com>

[smtp]
host = smtp.gmail.com
port = 587
username = JoeDoe@gmail.com
tls = true
```

Just as in case of HTTP, you must set username, but must not set password here to use the extension, in other cases it will revert to the default behaviour.

Usage

Configure the repository as above, then just pull and push (or email) You should be asked for the password only once (per every username + remote_repository_url combination).

8.5 pbranch

Patch Branches (pbranch) is a way to develop a series of patches for submission into a main repo. It is based on topic branches, one per patch, and is thus highly suitable for collaborative and/or long-term patch development and maintenance.

A detailed manual can be found online.

It adds a number of commands which can be listed with **hg help pbranch**:

```
pbackout       - backs out the current patch branch (undoes all its changes)
pdiff          - prints the final diff for the current or given patch branch
peditmessage   - edit the patch message
pemail         - send patches by email
pexport        - exports patches
```

```
pextdiff        - combines pdiff and extdiff
pgraph          - print an ASCII art rendering of the patch dependency graph
pmerge          - merge pending heads from dependencies into patch branches
pmessage        - print the patch message(s)
pnew            - start a new patch branch
pstatus         - print status of current (or given) patch branch
reapply         - reverts the working copy of all files touched by REV to REV
```

TortoiseHg will provide a new task tab that shows the patch dependency graph.

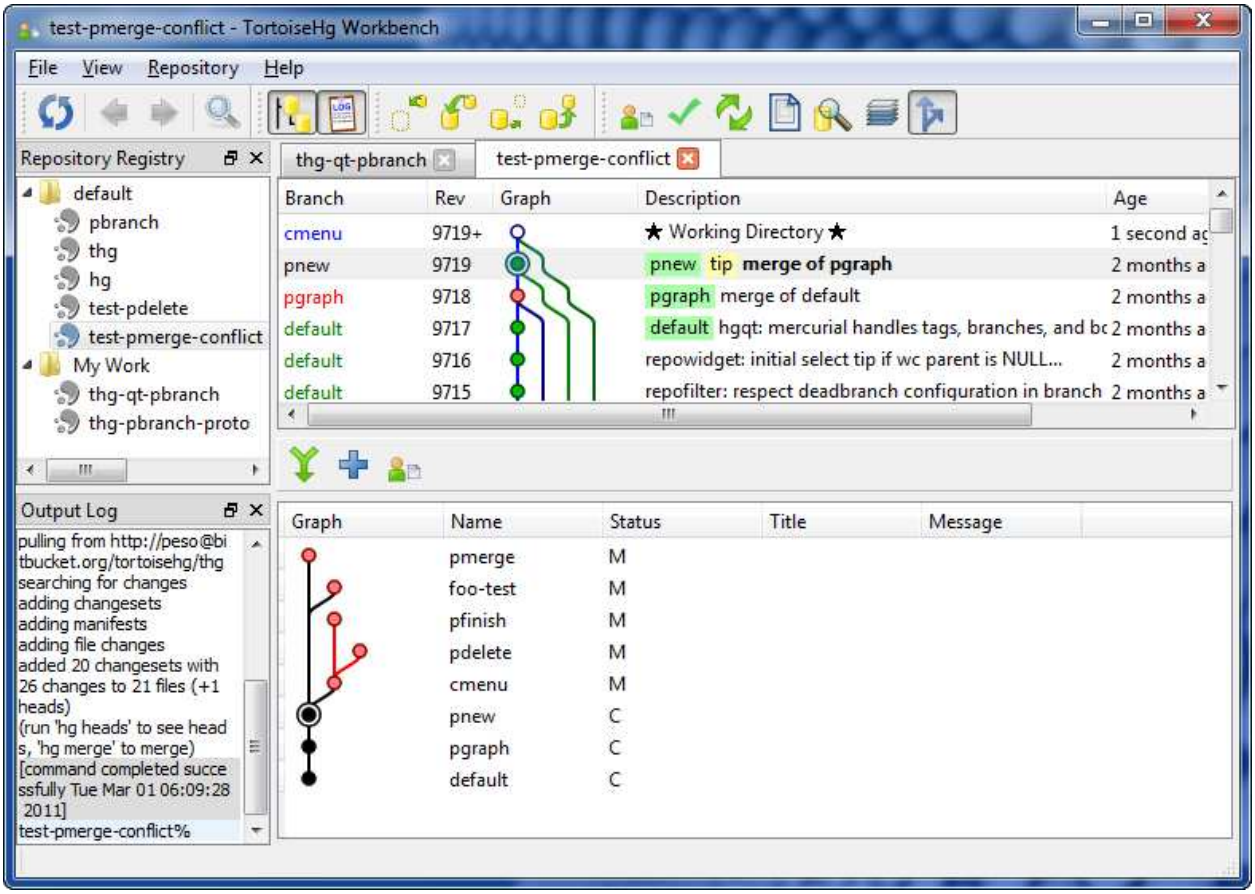


Figure 8.1: Pbranch task tab

Installation

As this extension is not installed with TortoiseHg, you have to download it from http://bitbucket.org/parren/hg-pbranch. Be sure to dowload the right one according to the Mercurial version included with TortoiseHg (see the wiki page on the download site). To test the use of this plugin, you can specify it on the Mercurial command line like this:

```
hg --config "extensions.pbranch=C:\path\to\pbranch.py" pstatus
```

You may want to add it to your Mercurial.ini or a repository's hgrc like this:

```
[extensions]
pbranch = C:\path\to\pbranch.py
```

If you do this, you can omit the –config command-line option.

CHAPTER

NINE

USE WITH OTHER VCS SYSTEMS

This chapter describes the three most popular Mercurial extensions for interoperating with *foreign* VCS systems. See also Repository Conversion

9.1 Perfarce (Perforce)

- Perfarce home page.
- Mercurial for Perforce users

This extension modifies the remote repository handling so that repository paths that resemble:

```
p4://p4server[:port]/clientname
```

cause operations on the named p4 client specification on the p4 server. The client specification must already exist on the server before using this extension. Making changes to the client specification Views causes problems when synchronizing the repositories, and should be avoided.

Five built-in Mercurial commands are overridden.

outgoing:

```
If the destination repository name starts with p4:// then this
reports files affected by the revision(s) that are in the local
repository but not in the p4 depot.
```

push:

```
If the destination repository name starts with p4:// then this
exports changes from the local repository to the p4 depot. If no
revision is specified then all changes since the last p4 changelist
are pushed. In either case, all revisions to be pushed are foled
into a single p4 changelist.  Optionally the resulting changelist is
submitted to the p4 server, controlled by the --submit option to
push, or by setting **perfarce.submit** to True.  If the option
**perfarce.keep** is False then after a successful submit the files
in the p4 workarea will be deleted.
```

pull:

```
If the source repository name starts with p4:// then this imports
changes from the p4 depot, automatically creating merges of
changelists submitted by hg push.  If the config option
**perfarce.keep** is False then the import does not leave files in
the p4 workarea, otherwise the p4 workarea will be updated with the
new files.
```

incoming:

```
If the source repository name starts with p4:// then this
reports changes in the p4 depot that are not yet in the local
repository.
```

clone:

```
If the source repository name starts with p4:// then this
creates the destination repository and pulls all changes from
the p4 depot into it.
```

The **perfarce.tags** configuration option determines whether perfarce tries to import Perforce labels as Mercurial tags.

TortoiseHg Integration

When the perfarce extension is enabled, it adds a **start revision** configurable option to the clone tool, and a **P4** toolbar button to the sync tool.

The toolbar button performs the p4pending operation. It detects pending Perforce changelists that have been "push"ed to your Perforce client but have not been submitted, or have not been pulled back. This opens the pending changelist dialog so that you can view these pending changelists and either submit or revert them. If Perforce fails the submit because your files are out of date, you must revert the changelist, pull from Perforce, merge, then push again.

Installation

Perfarce comes bundled with TortoiseHg Windows installers, so you enable perfarce by simply adding it to your Mercurial.ini or a repository's hgrc like this:

```
[extensions]
perfarce=
```

Note: The perfarce extension has been known to not work together with hgsubversion, so if you plan to use both extensions they should be enabled locally on the repositories that require them.

9.2 hgsubversion (SVN)

- hgsubversion home page
- hgsubversion Extension wiki page
- Working with Subversion Repositories

hgsubversion, as it's name implies, allows you to use Mercurial as a client to a Subversion server. It can also be used to do straight conversions of Subversion repositories into Mercurial.

Installation

TortoiseHg Windows installers come with the python-svn bindings that hgsubversion requires, so one only needs to clone the hgsubversion repository to your local computer:

```
hg clone http://bitbucket.org/durin42/hgsubversion/ C:\hgsvn
```

Then enable the extension in your Mercurial.ini file, specifying the hgsubversion folder inside the cloned repository:

```
[extensions]
hgsubversion = C:\hgsvn\hgsubversion
```

You can verify that worked by typing **hg help hgsubversion**

See the hgsubversion wiki for details of use. We recommend an hgsubversion version of at least 1.2.1 with Mercurial 1.8.

> **Warning:** When cloning a Subversion server, it is highly recommended to clone only the first few revisions then pull the rest. The failure behavior of the clone command is to delete the incomplete clone, while pull is much more forgiving.

TortoiseHg Integration

Imported Subversion changesets will display the original Subversion checkin number in the Repository Explorer browser.

9.3 hg-git (git)

- hg-git home page
- hg-git Extension wiki page
- Mercurial for Git users

hg-git, as its name implies, allows you to use Mercurial as a client to a git server. It can also be used to do straight conversions of Git repositories into Mercurial.

Installation

TortoiseHg Windows installers come with the python-git bindings (named dulwich) that hg-git requires, so one only needs to clone the hg-git repository to your local computer:

```
hg clone http://bitbucket.org/durin42/hg-git/ C:\hg-git
```

Then enable hggit in your Mercurial.ini file:

```
[extensions]
hggit = C:\hg-git\hggit
```

You can verify that worked by typing **hg help hggit**

Current versions of TortoiseHg include dulwich version 0.7.0. We recommend to use hg-git version 0.2.6 with this version of dulwich and Mercurial 1.8.

See the hggit documentation for details of use.

Beware the 'incoming' command appears broken when speaking with git repositories, and 'outgoing' does not show much useful info. So you are restricted to simple push and pull commands, which is common with Mercurial extensions that speak to external revision control tools.

CHAPTER

TEN

FREQUENTLY ASKED QUESTIONS

What is TortoiseHg?

A Windows shell extension for the Mercurial revision control system, similar to the Tortoise clients for Subversion and CVS. It also includes a thg application for command line use on many platforms.

What comes included in the TortoiseHg binary installer for Windows?

Mercurial, kdiff3, TortoisePlink bonus extensions: hgfold, perfarce, mercurial-keyring. fixfrozenexts, python-svn for hgsubversion and convert extensions, and dulwich for hg-git use. See `extension-versions.txt` in the TortoiseHg folder for more details on the exact versions packaged.

Is Mercurial on Windows compatible with the index service and virus scanners?

No. Like TortoiseSVN, we recommend to turn off the indexing service on the working copies and repositories, and exclude them from virus scans.

How can I get translations for the Explorer context menu?

The available translations were stored by the installer under `C:\Program Files\TortoiseHg\i18n\cmenu`. Select the locale you would like to use, double-click on it, and confirm all requests.

How do I do merges and arbitrary version checkouts?

Merges and updates are intended to be done within the **Workbench**, using changeset context menus

Why can't I connect to an ssh server (remote: bash: <server name>: command not found)?

TortoisePlink (and basic Plink) will try to use the **Host Name** configured in Putty under the **Default Settings**. It adds this host name to its command line parameters, causing the hostname to be specified twice, causing this particular error. Clearing the host name from the **Default Settings** is a possible workaround.

How can I use tool X as my visual diff tool?

Since version 1.0, TortoiseHg should autodetect most popular visual diff tools and make them available for selection from the **Visual Diff Tool** item in the settings tool.

How is TortoiseHg configured?

TortoiseHg gets configuration settings from two systems.

1. **The Mercurial configuration system, which is three-tiered** (a) Site-wide `Mercurial.ini` in `%ProgramFiles%\TortoiseHg`

   (b) Per-User `Mercurial.ini` in `%UserProfile%`

   (c) Per-Repository `hgrc` in `repo-root\.hg`

2. **The folder %APPDATA%\TortoiseHg:** (a) File `thg-reporegistry.xml` holds the content of the RepoRegistry.

   (b) File `TortoiseHgQt.ini` contains the settings for application state (window positions, etc).

These are some of the configurables that are stored in the Mercurial configuration system.

```
[tortoisehg]
vdiff = vdiff
editor = gvim
tabwidth = 4
longsummary = True
authorcolor = True
authorcolor.steve = blue
```

Where do TortoiseHg extensions look for external Python modules on Windows?

TortoiseHg includes an entire Python distribution bundled up as DLLs. The standard library modules are all in the `library.zip` file in `C:\Program Files\TortoiseHg`.

If you try to use an extension that imports a non-standard Python module, you will find that the extension will fail to load because it can't find the module. For example the ReviewBoard extension imports the simplejson module, which is not part of the standard Python distribution.

In order to make it work you need to add a couple of lines to the top of the extension's .py file, before the line that imports the foreign module:

```
import sys
sys.path.append(r'C:\path\to\module')
```

Note that this will not work for modules distributed as .egg files; the supplied path must contain the module's .py or .pyc files.

If you have many extensions and/or hooks that all share the same Python package, you can create an extension which explicitly modifies sys.path for all the others. Simply name the extension such that it is loaded first (alphabetically). Something like:

```
[extensions]
00setSysPath = C:\path\to\setsyspath.py
```

CHAPTER

ELEVEN

DEBUGGING

11.1 Dialogs

Stderr is being captured to a buffer that is being inspected at program exit. If any serious errors (tracebacks, etc) are found in the stderr buffer the entire contents are sent to the bug report tool so the user can (should) report a bug. If you suspect there are errors that are not being reported, you can set the environment variable **THGDEBUG** to any value to disable the stderr buffering.

If you have a bit of Python knowledge, you can also use:

```
thg --debugger <command>
```

To disable the forking behavior of thg, you can either set an environment variable **THG_HGTK_SPAWN**, or add the command line parameter '–nofork'.

11.1.1 Windows

To debug the changelog viewer, for instance, enter these commands into a **cmd.exe** window, while inside the repository:

```
set THGDEBUG=1
thg --nofork log
```

11.1.2 Linux/MacOSX

To debug the changelog viewer, for instance, enter these commands into your shell window, while inside the repository:

```
export THGDEBUG=1
thg --nofork log
```

11.2 Shell Extension

The debugging mechanisms depend on your platform.

11.2.1 Windows

See also http://msdn.microsoft.com/en-us/library/cc144064(VS.85).aspx for some info bits about Running and Testing Shell Extensions on Windows

The **DbgView** tool from the SysInternals suite will capture debug messages from the shell extension. However, the shell extension does not emit debugging info by default. It must be enabled by setting the registry key defined in `win32/shellext/DebugShellExt.reg` in the TortoiseHg source repository. You can double-click on this file to load the key into your registry.

Another option is to exit the **ThgTaskbar** system tray application and start it from the command line. It will emit some debug information to the console.

11.2.2 Nautilus

Debugging is done via the environment variable DEBUG_THG

- to test in a separate process:

```
DEBUG_THG=Ne TMPDIR=/tmp/anydir/ --no-desktop nautilus [path]
```

- to test in the main instance:

```
nautilus -q
DEBUG_THG=NOe nautilus
```

- permanent debugging, set DEBUG_THG in a file which is read on session start (~/.profile, ~/.xprofile)

Upper case characters in DEBUG_THG specify modules. Only *O* and *N* for *OverlayCache* and *Nautilus*, respectively, are supported module names. Lower case characters imply parts. Only *e* is supported, implying *error* messages.

To restart nautilus, chose either

1. killall nautilus (the session restarts nautilus automatically, stdin and stdout go to ~/.xsession-errors)
2. nautilus -q; nautilus (stdin and stdout are on the console)

CHAPTER

TWELVE

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

INDEX