e-monotone

A pcl-cvs like interface to monotone

Willem Rein Oudshoorn

An emacs front end for monotone, largely inspired on pcl-cvs. This document describes version 0.4

Table of Contents

1 Introduction 1

2 Installation 2
- 2.1 Compatibility 2
- 2.2 Configuration 2

3 Using e-monotone 3
- 3.1 The main window 3
  - 3.1.1 Filtering 4
  - 3.1.2 Examining files 4
  - 3.1.3 Reverting 4
  - 3.1.4 Adding/Dropping 4
- 3.2 Committing 5

4 Advanced Uses 6
- 4.1 Marking 6
- 4.2 Diffing revisions 6
- 4.3 Revision graphs 6
- 4.4 Configuration 7
- 4.5 Types of revision graphs 7
  - 4.5.1 Reduced display 7
  - 4.5.2 Branch crossings 7
  - 4.5.3 Incoming branches 7
  - 4.5.4 Major branch crossings 7
  - 4.5.5 Annotate display 8

5 BUGS 9

6 TODO 10

Index 11

1 Introduction

This package is a simple emacs interface to monotone. In its use it is somewhat similar to the pcl-cvs front end for CVS. However pcl-cvs is a mature package and this one is not, so expect a lot of rough edges, bugs and not a lot of advanced features.

For people who do not know pcl-cvs, this mode will allow you to manage an existing monotone project. It will

- Show a listing of all the files in the project, including their state.
- Allow you to show a diff of an edited file against the repository
- Allow you to commit the whole tree and specify a commit message.
- Allow to undo file changes, revert the file to version in the monotone database
- Add and Drop files.

What it will not do yet, but is desirable for the future

- Branching, allow to switch to a different branch.
- Synchronization, synchronize your changes with another database
- Merging

What I do not envision this mode to do is

- Create a new project
- Checkout a new project
- etcetera

Although, it might be added a la gnus's server configuration.

2 Installation

Installation is pretty straightforward, copy the elisp files, '`e-monotone.el`', '`e-monotone-a.el`', '`e-monotone-wami.el`', '`e-monotone-certs.el`' and '`e-monotone-graphs.el`' to a location in the emacs load path, and put the following in your '`.emacs`' file

```
(load "e-monotone.el")
```

If emacs complains during startup that it can not fine '`e-monotone-a.el`' or any other e-monotone file, the files are not installed the emacs load path. The load path can be changed with

```
(add-to-list 'load-path (expand-file-name "/path/to/e-monotone"))
```

2.1 Compatibility

At the moment it is only tested with monotone version 0.23. It uses only a few monotone commands, but these includes `automate inventory` and `automate heads`. So if the output format is changed it probably will not work. Also for performance reasons, it will use `db execute` to get all the certificates. This does not work with version 0.26 because the database has changed. Most things will still work, but most noticably, the graph drawing is not as nice with version 0.26.

2.2 Configuration

Emacs need to be able to find the `monotone` executable. Default it tries to find an executable with the name `monotone` or `mtn` in the `PATH`. If this does not work you can tell emacs which executables to use by putting

```
(setq monotone-configuration '(("MT" "/path/and/monotone-executable-pre-0.26-name" t)
                               ("_MTN" "/path/and/monotone-executable-0.26-or-higher"
```

in your '`.emacs`' file. If you are on windows, you can use dos style paths but don't forgot to quote the slash. So a path should look like `f:\\path\\monotone.exe"`.

Similar for viewing the graphs in emacs you might want to set the path to the `dot` executable. If `dot` is not found add

```
(setq dot-cmd "/path/and/dot-executable-name")
```

The `dot` program is part of graphviz and is used to convert graphs pictures. The `dot` program needs to be able to convert to '`.png`' files.

3 Using e-monotone

3.1 The main window

The first thing to do is to have a monotone project. Now lets assume this project lives in the directory '`~/src/em`'. Now to start the e-monotone you use the command *M-x monotone-examine*. This will prompt for a directory in the usual emacs way. So just supply the directory above, '`~/src/em`' and you end up with a screen that roughly looks like:

```
directory ~/src/em
revision  3f70e5cd7cee3b080767aa447354dcb16bcff2ef
heads      3f70e5cd7cee3b080767aa447354dcb16bcff2ef

  branch "nl.xs4all.ironhead.emacs-monotone"
database "/home/woudshoo/monotone/configuration.db"
     key ""

Working copy based on head, and there are pending changes

edited:                0    added:                 1    unknown:   16
renamed-and-edited:    0    added-and-missing:     0    ignored:    3
renamed:               0    dropped:               0
unchanged:             2    missing:               0    TOTAL:     22

Currently displaying (added added-and-missing dropped edited renamed missing)


       added  e-monotone.texi
```

Now there a few sections in in the screen, the first two lines indicate on which revision our current source is based, and what the monotone-database thinks the most up to date revisions are. In this case, they are equal, so we are working on the head revision. This information is also displayed in the 8-th line, which is the summary line. The summary line displays the most important information, in the example it is

```
Working copy based on head, and there are pending changes
```

and here we can see that we are based on head, so committing will not create an additional head and there are local changes.

Just above the summary line we have some general information, the branch we work on, the database that is used and the default key that is used. Actually it is just the content of the '`MT/options`' file.

Below the summary line we display some statistics of the project. Here we see that the project contains 22 files, of which 19 are not under version control (the unknown and the ignored files) one is newly added, and 2 are already under version control but not changed.

Below the statistics section we see the file listing. In this example it shows only one file, the newly added '`e-monotone.texi`' file. It shows only one file, because the file listing is

filtered. At the moment it shows only interesting files. The buffer explains which files are displayed by the line:

```
Currently displaying (added added-and-missing dropped edited renamed missing)
```

So it does display the added file, but not the 16 unknown and not the 3 ignored files.

3.1.1 Filtering

As seen in this example, the list of files is filtered. When *M-x monotone-examine* is invoked it starts with displaying only the interesting files. The interesting files are files which according to monotone have one of the following states `added, dropped, renamed, edited, missing, added-and-missing`. This is done to avoid cluttering the buffer all the uninteresting files. However it is very easy to change the filtering. If you want to see all files just press the key `a` for all and you will see all files listed, in this case 22. Pressing *i* will bring you back to the list of interesting files. There is one other predefined filter, accessed by *k*, the known file list. This displays all files that are under version control.

It is also possible to show/hide per category.

sa	Toggles the display of the added files
sd	Toggle the display of dropped files
si	Toggles the display of ignored files
sm	Toggle the display of missing files
se	Toggle the display of edited files
su	Toggle the display of unchanged files
sr	Toggle the display of renamed files
s?	Toggle the display of unknown files

3.1.2 Examining files

When the cursor is positioned on file line the ⟨RET⟩ will open this file in a new buffer. If the file was edited you can use the `=` key to open a buffer containing the difference between the current content and the content of the file as it was when this version was checked out.

3.1.3 Reverting

After you have carefully considered the output of `=` and decide that you do not want this change, for example because you have only added temporary debugging statement, you can use *U* in the monotone buffer, with the cursor on the line of the file, to revert this file back to the original state.

3.1.4 Adding/Dropping

In the monotone buffer the keys *A* and *D* can be used to add respectively drop files from monotone.

3.2 Committing

Finally we want to commit. By pressing `C` a buffer is opened that contains a description of all the changes that are being committing. All the lines in this buffer start with `MT:`. Just before the actual commit takes place all the lines starting with `MT:` are removed and the text that is left is used as the commit message. The user initiates the commit by pressing `C-c C-c` in the commit buffer. So normally you press `C` in the monotone buffer, add some lines in the commit buffer and finish it off with `C-c C-c`. If at any time you want to remove the comment lines, you can do that by `C-c C-r`. If you do not want to see these lines at all you should start by using `c` instead of `C`. Using the lowercase version is equivalent with the capital version except that the commit buffer starts out empty.

4 Advanced Uses

4.1 Marking

In the monotone buffer you can mark files by pressing `m` and unmark them with `u`. After marking the commands that normally work on a single file will work on the marked collection. This allows you to Add, Delete, Revert or Commit only a selected number of files.

Note that marking and displaying are independent. So it is possible to have a lot of marked files and not see them flagged in the monotone buffer. Fortunately e-monotone will ask for confirmation when operating on the marked files.

The key `M-u` will unmark all marked files.

4.2 Diffing revisions

Pressing `=` on the revision ids in the top of the buffer will present a diff between that revision and your working copy.

Presssing `t` in the monotone buffer on a line containing a file will show the annotated version.

The annotated file is by default displayed with on the left hand side a shortened revision hash and on the right hand side the content of the file.

The display can be changed in several ways,

`ss` Shows the shortened revision hashes, this is the default.

`sf` Show the full length revision hash.

`sa` Show the value of the author certificate

`sd` Show the value of the date certificate

Furthermore, the way the lines are coloured can be changed, the default is to more or less assign a random colour to a revision. By key sequence `cd` will colour the revisions by date. Pressing `cr` will revert to the random colouring of revisions.

In the monotone and annotate buffer pressing `W` show a revision graph rendered by `dot`.

4.3 Revision graphs

A monotone database typically holds hundredths or thousands of revisions, so a displaying a complete revision graph is a bit useless. The graph needs to be reduced to a managable size.

A simple way of reducing the graph is to pick an interesting node, like the current revision, and only show a small neighborhood of this node.

This is of course not the only way to reduce a graph, but to keep it managable in monotone all graphs displayed are limited by the number of nodes displayed. The numebr of nodes that is displayed is increased by the `+` key and reduced by the `-`. Because the image displayed has a fixed size this will result in some kind of zooming effect.

4.4 Configuration

There are two important variables that you can configure for the displaying of the revision graphs,

`monotone-preffered-branches`
: This variable contains a list of regular expressions. These expressions are used when a revision has multiple branch certificates. If a revision has multiple branch certificates and one of branch names matches one of the regular expressions in the preffered-branches variables it will assume the matched branch name for coloring.

`monotone-wami-node-displayed-certs`
: This variable determines which certificates are displayed in the diagrams. It is a list of certificate names to display. A list element is either a string, in which case the value is displayed as such, or a cons pair. The first element of the cons pair is a certifacte name and the second element a functions which takes as argument the branch value and returns the string to display instead.

4.5 Types of revision graphs

Beside just taking a number of neighbouring nodes e-monotone has the ability to modify the displayed graph in interesting ways.

4.5.1 Reduced display

A basic reduction step is to remove all nodes that have exactly one parent and exactly one child. Removing these nodes will leave the topology of the graph intact and allow you to display more of the neighbourhood of the interesting node than otherwise would be the case. The displayed graph will display the edge that previously contained the node blue, so you have an indication that some revisions are ommited along the blue arc.

4.5.2 Branch crossings

This reduction steps try to display the interaction between branches by mostly ignoring what happens inside a branch. The reduction tries to remove as many nodes as possible, only keeping those who either have a parent in a different branch or a child in a different branch. Also in order to keep an overview, head nodes and root nodes are not deleted.

4.5.3 Incoming branches

This concentrates on what is merged in from other branches in the current branch. Here current branch is determined by which branch the intereseting revision belongs to. The nodes that are kept are the nodes in the current branch who have a parent in a different branch. Also nodes in a branch different from the current branch who have a child in the current branch are kept. As usual the heads and root node of the current branch are not discarded.

4.5.4 Major branch crossings

This needs to be documented and made configurable.

4.5.5 Annotate display

If the graph is generated from the annotate buffer the graph displayed will display the current revision plus all revisions that are mentioned in the annotate buffer.

5 BUGS

There a plenty bugs I know of,

- Cursor position is not well maintained when buffer is redisplayed
- Lots of file states are not handled correctly, for example renaming
- Font lock coloring of the commit buffer is wrong
- After add/drop/revert the file status displayed may be incorrect. Work around, regenerate with *g*
- Many many more

6 TODO

A lot :-)

Index

(Index is nonexistent)