



Hashes with History

Alberto Manuel Simões
 ambs@di.uminho.pt

Hashes are nice data structures but lack on the possibility of a revision history. Many times I have wanted to replace a value in a hash but save the previous one. While I could implement this with hashes of arrays, that has persistence problems: I can store it, but every time I need a value, I must retrieve the entire object.

The Tie::Cvs module ties hashes to Concurrent Versions System (CVS) trees, thus enabling automatic (well, almost) history control.

I define a CVS repository when I create the hash. For each key, I create a file in the CVS repository. Each time a key gets a new value, I check-in the new value as a file in the CVS tree. If I delete a key, I rollback to the latest version available.

I started writing this module to give revision control to a home-made wiki system. Basically it used a Berkeley DB file where keys were the page names and values were the page contents (a big string). Instead of creating my own versioning system with a Berkeley DB I changed to CVS and Tie::Cvs.

How Ties Work

Writing a tied object is easy: I just need a set of methods to connect the tied object with the hashes I'm tying. The most important methods are:

TIEHASH is called when I use the tie() function. Its first argument is the name of the class I use, and remaining arguments are passed in the same order I passed them to the tie() function.

For my particular object, when I tie a hash with a CVS tree I need to check if the tree exists. If it doesn't, I initialize a new one. If it exists, I check out a fresh copy.

FETCH is called whenever I access an element from the hash. If the hash is tied to an object, perl

calls this method on that object passing the key to be fetched.

For Tie::Cvs I must have an updated version of the CVS tree checked-out somewhere. With that, to fetch a data value based on the key is simply to return the contents of the file with the key name.

STORE is called when I store an element on the hash. perl calls this method on the tied object, passing the key and the value to be stored.

To store a new value on Tie::Cvs I must check if that key already exists: if not, I have to create a file with the key name and add it to the CVS repository; if it exists, I need to check it in.

DELETE is called on the tied object when I use the delete() command on a hash. Perl passes the key of the element to delete.

When deleting a key I must check its version. If it is the first, I remove the file from the repository. If not, I roll-back the file contents to the previous version.

EXISTS is called when I use the exists() command on the hash. For Tie::Cvs this method is as simple as to check if the file exists in the CVS repository.

Implementation Details

The following subsections each go over the tie methods. We've simplified some implementation details in some cases.

Tie Initialization

To tie a hash to Tie::Cvs, I use the tie command and give it a repository root directory.

```
tie %hash, "Tie::Cvs", "/cvs/root/";
```

Perl calls **TIEHASH** where first argument is the class name, Tie::Cvs, and second is the path to the CVS repository.