Distributed revision control for CCP4

Ville Uski

March 1, 2013

Contents

1 Introduction

In this report, I briefly discuss how to use a distributed version control system (DVCS) to maintain the CCP4 source repository. This would include converting the old CVS repository under the new system with full revision history. Earlier versions of this document compare three DVCSs, namely Bazaar, Mercurial and Git, but this one focuses on Bazaar only.

2 Data migration from CVS

2.1 Conversion

When converting the old CVS repository to a Bazaar repository, there are various options. For example, the following:

1. 'Top-skim', i.e., start the new repository from the latest revision of each module in the CVS repository. Earlier versions can still be obtained from the CVS repository, which should be made read-only.

2. Trunk only. Taking only the main development line from each module. This saves some space but still retains the revision history for the trunk.

3. Converting everything, including all tags and branches.

The first option is the easiest, and will work for every module. The latter options seem to work for all the other modules except ccp4. If that one is to be converted with full revision history, then some debugging will be required.

Let the environment variable CVSROOT point to the repository on the CVS server:

```
$ export CVSROOT=ehl25977@ccp4t.dl.ac.uk:/ccp4/ccp4null/repository
```

where ehl25977 is my username on the server. In the following, I'll discuss how to create a Bazaar repository under `localhost:/ccp4mirror/bzrroot`.

To create a top-skim version of the **pointless** module, one could proceed as follows:

```
$ mkdir -p /ccp4mirror/bzrroot
$ cd /ccp4mirror/bzrroot
$ cvs checkout pointless
$ cd pointless
$ rm -rf CVS
$ bzr init
$ bzr add
$ bzr commit
```

To convert the repository with full revision history (options 2 and 3), one requires a local copy of the CVS repository. This can be created using rsync:

```
$ mkdir /ccp4mirror/cvsroot
$ rsync -av -e ssh --delete $CVSROOT/ /ccp4mirror/cvsroot
```

So the local CVS repository will be under /ccp4mirror/cvsroot.

In principle, the conversion is accomplished simply with the help of one of the Bazaar plugins, namely, either **fast-import** or **cvsps-import**, which are both in the Ubuntu software repository. The usual recipe for converting with fast-import is this:

```
$ bzr fast-export-from-cvs source-repository project.fi
$ bzr fast-import project.fi project.bzr
```

As it turns out, neither of the commands work out of the box (the same was true for cvsps-import). The first command is just a wrapper for the cvs2bzr tool, which comes with the cvs2svn package. Using the tool directly works:

```
$ mkdir /ccp4mirror/bzrroot
$ cd /ccp4mirror/bzrroot
$ cvs2bzr --username=cvs2bzr \
```

```
--dumpfile=dumpfile.fi /ccp4mirror/cvsroot/pointless
```

You may need to add the `--retain-conflicting-attic-files` if some files appear both in and out of the CVS Attic. The second command (`bzr fast-import`) did not work due a known bug which has recently been fixed. So upgrading the fastimport plugin to the latest version resolved the problem. The latest version is obtained from Launchpad:

```
$ bzr branch lp:bzr-fastimport fastimport
```

If this is run in the ~/.bazaar/plugins directory, then Bazaar will use the new plugin.

After installation, I ran the command

```
$ bzr fast-import dumpfile.fi pointless
```

which creates a subdirectory `pointless`:

```
$ ls -la pointless
total 36
drwxr-xr-x 7 ville ccp4  4096 2011-09-29 18:51 .
drwxr-xr-x 7 ville ccp4  4096 2011-09-29 18:51 ..
drwxr-xr-x 4 ville ccp4  4096 2011-09-29 18:51 .bzr
drwxr-xr-x 3 ville ccp4  4096 2011-09-29 18:51 poinless-1_6_4-local
drwxr-xr-x 3 ville ccp4  4096 2011-09-29 18:51 pointless-1_5_22-local
drwxr-xr-x 3 ville ccp4  4096 2011-09-29 18:51 TAG.FIXUP
drwxr-xr-x 5 ville ccp4 12288 2011-09-29 18:51 trunk
```

Here, the directory .bzr includes the shared repository, whereas the other directories contain individual branches. 'trunk' denotes the main development branch and that directory also includes a working tree. The other branches don't have a working tree by default, but these can be created running `bzr checkout` inside each branch.

Before using the fast-import command, it's possible to apply a fast-import-filter to the dump file to include or exclude files and directories. cvs2bzr can be given an additional option `--trunk-only`, if only the main development line is to be converted.

2.2 Repository layout

As Bazaar does not (yet) support multiple (nested) repositories (Git and Mercurial do), we would probably set up the server such that each module (subdirectory) in the present CVS repository forms an independent repository under a root directory (`bzrroot`) on the server.

So the module pointless, for instance, will have a repository in the directory `pointless/.bzr`. This repository will be *shared* by all the branches of pointless

on the server. Each branch will have a subdirectory under `pointless` and these branches can be nested in arbitrary ways.

It would be possible to have just one repository shared by all the modules which would now be branches of the same 'supermodule'. This, however, is not the preferred way for Bazaar, as the primary benefit of a shared repository is that the branches share a common ancestry.

2.3 Running a central Bazaar server for CCP4

Bazaar operates over HTTP, FTP or SFTP, so it's possible to set up a 'dumb' server relying on these protocols. There is an optional 'smart' server that can be invoked over SSH or http (making use of Apache), from inetd, or in a dedicated mode. The smart server is said to provide a better performance, so here I'll discuss setting up a smart server for CCP4. I'll focus on using SSH to access it, as it requires no special configuration on the server. The other methods may be covered in later versions of this document.

The server can be accessed using the bzr+ssh method (replace MODULENAME with the name of the module, e.g., pointless and BRANCHNAME with the branch name). For example:

```
$ bzr log bzr+ssh://$BZRROOT/MODULENAME/BRANCHNAME
```

where I have set:

```
$ export BZRROOT=ehl25977@ccp4t.dl.ac.uk/ccp4/ccp4null/bzrroot
```

The variable has no special meaning in Bazaar, so I just use it for convenience. At the moment, Bazaar is not installed on the server, except in my home directory, so one should also set:

```
$ export BZR_REMOTE_PATH=/home/ehl25977/bin/bzr
```

Read and write permission to each branch are set up using the `bzr_access` script. Authentication is based on usernames and performed by SSH. First, an administrator must generate a private/public key pair for each user who will be accessing the repository. The details are explained in [3], Sec. 8.2.

2.4 Creating a new module on the server

To start a completely new module on the server ccp4t.dl.ac.uk, say, **new-mod**, one should first set path to the bzr executable on the server:

```
$ export BZR_REMOTE_PATH=/home/ehl25977/bin/bzr
```

and (for convenience) the path to the root directory on the server:

```
$ export BZRROOT=ccp4null@ccp4t.dl.ac.uk/ccp4/ccp4null/bzrroot
```

Locally, one should also set the current user:

```
$ bzr whoami "Ville Uski <ville.uski@stfc.ac.uk>"
```

Next, initialize the shared repository on the server:

```
$ bzr init-repo --no-trees bzr+ssh://$BZRROOT/new-mod
```

This will ask for a password for the user ccp4null. The `new-mod` directory will be created if it doesn't exist.

Assuming there is a stack of code locally under directory `new-mod`, we should first bring that under revision control:

```
$ cd new-mod
$ bzr init
$ bzr add
$ bzr commit -m "Initial revision"
```

and then copy it to the server:

```
$ bzr branch . bzr+ssh://$BZRROOT/new-mod/trunk
```

This creates a branch `trunk` without a working tree. The working tree can be created using `'bzr checkout'` on the server, but there is no particular reason to do this. The remaining task is to set the correct file permissions on the server, like this:

```
$ ssh ccp4null@ccp4t.dl.ac.uk chmod -R o-rwx /ccp4/ccp4null/bzrroot/new-mod
$ ssh ccp4null@ccp4t.dl.ac.uk chmod -R g+w /ccp4/ccp4null/bzrroot/new-mod
```

It should be possible to avoid the last step altogether by resetting the umask for ccp4null.

3 Using Bazaar

3.1 Centralized workflow

As Bazaar is designed to be very flexible, it supports many different workflow models [2]. I'll first discuss how to use it in the centralized manner like CVS. Distributed workflows can then be introduced gradually as deemed appropriate. This will assume that the server has already been setup under BZRROOT as discussed, and the local user is set as explained in Sec. 2.4.

Now we can check out individual projects like with CVS:

```
$ bzr checkout bzr+ssh://$BZRROOT/pointless/trunk pointless
```

which creates a local branch in the directory pointless, which is a replica of the trunk on the server. After working on the files in the directory, the changes are committed:

```
$ bzr commit
```

This will commit changes to both the local and remote branches. This feature is called **bound branches** and is unique to Bazaar.

Commits can be done only locally by using `'bzr commit --local'` or unbinding the local branch (`'bzr unbind'`, the branches can be bound again using `'bzr bind'`). This is useful if working offline or simply to reduce a chance of a bad commit. It is also possible to have lightweight checkouts (`bzr checkout --lightweight`) which will copy the working tree but not the branch. Lightweight checkouts depend on access to the branch for every operation, and so don't support local commits. They are similar to CVS checkouts.

If the branch on the server has already been modified by another user, Bazaar will make sure that the local repository is up to date before making the commit. There is a chance to either merge the branches or create a new one on the server.

After some developer has committed changes to the server, other developers would normally update (`'bzr update'`) their checkouts. Any local changes will still need to be committed or discarded (`'bzr revert'`) before the update is complete.

When branches split apart and are finally merged back together, they form a graph of revisions. Such a graph is called a directed acyclic graph (DAG) as it shows directed relationships between parent and child revisions and has no loops.

3.2 Gatekeeper workflow

Gatekeeper is a person[1] who commits the changes to the 'main' branch, kept on the server. She is the only one with commit rights to the main branch. In this workflow model, there are also a number of developers who pull changes from that branch, push to their own independent branch and ask the gatekeeper to review their changes. If happy, the gatekeeper merges their changes into her integration branch and commits to the main branch. So the developers would not check out their branches, but clone them instead:

```
$ bzr branch bzr+ssh://$BZRROOT/pointless/trunk pointless
```

[1]The gatekeeper does not need to be a person, as some software tools can be used as automatic gatekeepers [5]. An automatic gatekeeper will run a series of tests which must be passed before accepting any changes to the main branch.

Furthermore, instead of updating their local branches, they would pull the changes:

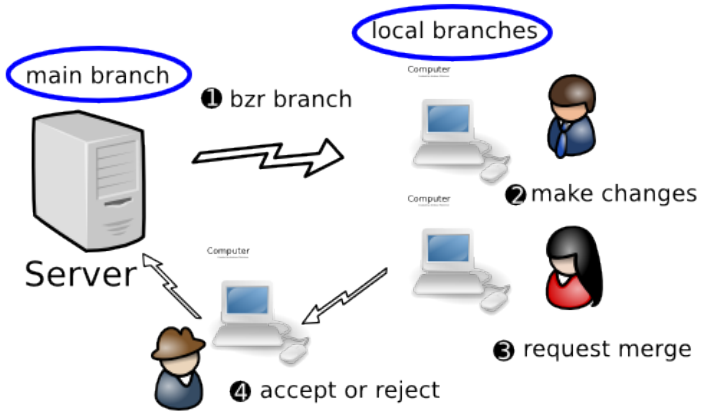
```
$ bzr pull bzr+ssh://$BZRROOT/pointless/trunk
```

This will only work, if the branches have not diverged. If it fails, one could merge instead:

```
$ bzr merge bzr+ssh://$BZRROOT/pointless/trunk
```

The location of the remote branch needs to be given only in the first pull/merge, after that it will be remembered.

Here is a picture from the User Guide [5] illustrating this workflow:



References

[1] *Why Switch to Bazaar?*, http://doc.bazaar.canonical.com/migration/en/why-switch-to-bazaar.html (accessed 20 Sep 2011)

[2] *Centralized Workflow Tutorial*, `http://doc.bazaar.canonical.com/latest/en/tutorials/centralized_workflow.html` (accessed 22 Sep 2011)

[3] *Bazaar System Administrator's Guide*, `http://doc.bazaar.canonical.com/latest/en/admin-guide/index.html` (accessed 22 Sep 2011)

[4] *Shared Repository Tutorial*, `http://wiki.bazaar.canonical.com/SharedRepositoryTutorial` (accessed 22 Sep 2011)

[5] *Bazaar User Guide*, `http://doc.bazaar.canonical.com/latest/en/user-guide/index.html` (accessed 22 Sep 2011)

[6] *Why Git is better than X?*, http://whygitisbetterthanx.com

[7] *Mercurial equivalents of CVS*, http://mercurial.selenic.com/wiki/CvsCommands

[8] *Handling repository events with hooks*, in Bryan O'Sullivan, *Mercurial: The Definitive Guide*, http://hgbook.red-bean.com/read/handling-repository-events-with-hooks.html

[9] *Shared SSH*, `http://mercurial.selenic.com/wiki/SharedSSH`, (retrieved 11 October 2011).