Setting up a CVS repository - the FreeBSD way

Stijn Hoop

stijn@win.tue.nl

$FreeBSD: release/8.4.0/en_US.ISO8859-1/articles/cvs-freebsd/article.xml 39632 2012-10-01 11:56:00Z gabor $

$FreeBSD: release/8.4.0/en_US.ISO8859-1/articles/cvs-freebsd/article.xml 39632 2012-10-01 11:56:00Z gabor $

This article describes the steps I took to set up a CVS repository that uses the same scripts the FreeBSD project uses in their setup. This has several advantages over a stock CVS setup, including more granular access control to the source tree and generation of readable email of every commit.

1 Introduction

Most of the open source software projects use **CVS** as their source code control system. While **CVS** is pretty good at this, it has its share of flaws and weaknesses. One of these is that sharing a source tree with other developers can quickly lead to a system administration nightmare, especially if one wishes to protect parts of the tree from general access.

FreeBSD is one of the projects using **CVS**. It also has a large base of developers located around the world. They developed some scripts to make management of the repository easier. Recently, these scripts were revisited and normalized by Josef Karthauser <joe@FreeBSD.org> to make it easier to reuse them in other projects. This article describes one method of using the new scripts.

To make the most use of the information in this article, you need to be familiar with the basic method of operation of **CVS**.

2 First setup

> **Warning:** It might be best to first perform this procedure with an empty test repository, to make sure you understand all consequences. As always, make sure you have recent, readable backups!

2.1 Initializing the repository

The first thing to do when setting up a new repository is to tell **CVS** to initialize it:

```
% cvs -d path-to-repository init
```

This tells **CVS** to create the `CVSROOT` administrative directory, where all the customization takes place.

2.2 The repository group

Now we will create the group which will own the repository. All committers need to be in this group, so that they can write to the repository. We will assume the FreeBSD default of `ncvs` for this group.

```
# pw groupadd ncvs
```

Next, you should chown(8) the directory to the group you just added:

```
# chown -R :ncvs path-to-your-repository
```

This ensures that no one can write to the repository without proper group permissions.

2.3 Getting the sources

Now you need to obtain the `CVSROOT` directory from the FreeBSD repository. This is most easily done by checking it out from a FreeBSD anonymous CVS mirror. See the relevant chapter in the handbook (http://www.FreeBSD.org/doc/en_US.ISO8859-1/books/handbook/anoncvs.html) for more information. Let us assume that the sources are stored in `CVSROOT-freebsd` in the current directory.

2.4 Copying the FreeBSD scripts

Next, we will copy the FreeBSD `CVSROOT` sources into your own repository. If you are accustomed to **CVS**, you might be thinking that you can just import the scripts, in an attempt to make synchronizing with later versions easier. However, it turns out that **CVS** has a deficiency in this area: when importing sources into the `CVSROOT` directory, it will not update the needed administrative files. In order to make it recognize those, you will need to checkin each file after importing them, losing the value of `cvs import`. Therefore, the recommended method is to simply copy over the scripts.

It does not matter if the above paragraph did not make sense to you—the end result is the same. Simply check out your `CVSROOT` and copy the FreeBSD files over your local (untouched) copies:

```
% cvs -d path-to-your-repository checkout CVSROOT
% cd CVSROOT
```

```
% cp ../CVSROOT-freebsd/* .
% cvs add *
```

Note that you will probably get a few warnings about some directories not being copied; this is normal, you do not need those.

2.5 The scripts

Now you have in your working directory an exact copy of the scripts that the FreeBSD project itself uses for their repository. A summary of what each file is used for is included below.

- `access` - this file is not used in the default setup. It is used in the FreeBSD project specific setup, where it controls access to the repository. You can remove this file if you do not wish to use this setup.
- `avail` - this file controls access to the repository. In this, you can specify groups of people that are allowed access to the repository, as well as disallow commits on a per-directory or per-file basis. You should tailor it to contain the groups and directories that will be in your repository.
- `cfg.pm` - this file parses your configuration, and provides the default configuration. You should *not* make changes to this file. Instead, put your configuration changes in `cfg_local.pm`.
- `cfg_local.pm` - this file contains all configurable parameters of the system. You should configure all sorts of settings here, such as where commit mail is send, on what hosts people can commit, and others. More information on this below.
- `checkoutlist` - this files lists all files under control of **CVS** in this directory, apart from the standard ones created by `cvs init`. You should edit this to remove some FreeBSD-specific files.
- `commit_prep.pl` - this script performs various pre-commit checks, based on whether you enabled them in your `cfg_local.pm`. You should not have to touch this.
- `commitcheck` - this script is invoked directly from **CVS**. It first checks if the committer has access to the specified part of the tree using `cvs_acls.pl`, and then runs `commit_prep.pl` for the various pre-commit checks. If those are OK, **CVS** will allow the commit to proceed. You should not have to touch this file.
- `commitinfo` - this file is used by **CVS** to determine which script to run before a commit—in this case `commitcheck`. You should not have to touch this file.
- `config` - the configuration file for this repository. You should change this as needed, but most administrators can probably leave the defaults. More information on the options that can be set here can be found in the **CVS** manual.
- `cvs_acls.pl` - this script determines the committers identity, and whether he/she is allowed access to the tree. It does this based on the `avail` file. You should not have to touch this file.
- `cvsignore` - this file specifies files that **CVS** should not checkin in the repository. You can edit this as you wish. More information about this file is available in the **CVS** manual.
- `cvswrappers` - this file is used by **CVS** to enable or disable keyword expansion, or whether a file should be considered binary. You can edit this as you wish. More information about this file is available in the **CVS** manual. Note that the `-t` and `-f` options do not work correctly with client/server **CVS**.
- `edithook` - this file is not used any more, but kept for historic reasons. You can safely remove this file.

- `editinfo` - **CVS** uses this file for editor overrides. FreeBSD does not use this functionality, as parsing the log message is done by `verifymsg` and `logcheck`. This is because the `editinfo` functionality does not work properly with remote commits, or ones that use the `-m` or `-F` options. You should not have to touch this file.
- `exclude` - this file lists regular expressions that are used by `commit_prep.pl` to determine files which cannot contain a revision header. In the FreeBSD setup, all files under revision control need to have a revision header (like $FreeBSD$). All filenames that match one of the lines in this file are exempted from this check. You should add expressions to this file as you checkin files that cannot have a revision header. For the purpose of installing the scripts, it may be best to exclude `CVSROOT/` from header checks.
- `log_accum.pl` - this is a script that takes the log message as provided by the `logcheck` script, and appends it to a log file in the repository for backup purposes. It also handles mailing out a message to an email address you provide (in `cfg_local.pm`). It hooks into **CVS** via `loginfo`. You should not have to touch this file.
- `logcheck` - this file parses the commit log message that committers provide, and attempts to sanitize it somewhat. It hooks into **CVS** via `verifymsg`. You should not have to touch this file.

  Note: This script depends on a local FreeBSD hack of **CVS**: this version reads the log message back in after this script has modified it. The stock version of **CVS** does not do this which makes `logcheck` unable to clean up the log message, although it is still able to check that it is syntactically OK. **CVS** 1.11.2 can be configured to have the same behaviour as FreeBSD's version by setting `RereadLogAfterVerify=always` in the `config` file.

- `loginfo` - this file is used by **CVS** to control where log information is sent; `log_accum.pl` hooks in here. You should not have to touch this file.
- `modules` - this file retains its traditional meaning in **CVS**. You should remove the FreeBSD modules from the stock version. You can edit this as you wish. More information about this file is available in the **CVS** manual.
- `notify` - this file is used by **CVS** in case someone sets a watch on a file. It is not used in the FreeBSD repository. You can edit this as you wish. More information about this file is available in the **CVS** manual.
- `options` - this file is specific to the FreeBSD version of **CVS**, and is also supported by the Debian version. It contains the keyword to expand in revision headers. You should alter this to match the keyword you specified in `cfg_local.pm` (if you use that feature, which is FreeBSD specific for now).
- `rcsinfo` - this file maps directories in the repository to template files such as `rcstemplate`. By default, FreeBSD uses one template for the whole repository. You can add others to this file if you wish.
- `rcstemplate` - this file is the actual template committers will see when they make a checkin. You should edit this to describe the various extra parameters you defined in `cfg_local.pm`.
- `tagcheck` - this files controls access to tagging in the repository. The stock FreeBSD version disallows tags with names of RELENG*, because of the release engineering process. You should edit this file as desired.
- `taginfo` - this file maps tag operations on repository directories to access control scripts such as `tagcheck`. You should not have to touch this file.
- `unwrap` - this script can be used to automatically "unwrap" binary files (see `cvswrappers`) on checkout. It is not used in the current FreeBSD setup because the functionality it hooks into does not work well with remote commits. You should not have to touch this file.
- `verifymsg` - this file maps repository directories to post processor scripts of log messages such as `logcheck`. You should not have to touch this file.

- `wrap` - this script can be used to automatically "wrap" binary files (see `cvswrappers`) on checkin. It is not used in the current FreeBSD setup because the functionality it hooks into does not work well with remote commits. You should not have to touch this file.

2.6 Customizing the scripts

The next step is to set up the scripts so that they work in your environment. You should go over all files in the directory and make your customizations. In particular, you might want to do edit the following files:

1. If you do not wish to use the FreeBSD specific features of the scripts, you can safely remove the `access` file:

   ```
   % cvs rm -f access
   ```

2. Edit `avail` to contain the various repository directories in which you want to control access. Make sure you retain the `avail||CVSROOT` line, otherwise you will lock yourself out in the next step.

   The other thing you can add in this file are committer groups. By default, FreeBSD uses the `access` file to list all its committers in, but you can use any file you wish. You can also add groups if you want (the syntax is specified at the top of `cvs_acls.pl`).

3. Edit `cfg_local.pm` to contain the options you want. In particular, you should take a look at the following configurable items:

   - `%TEMPLATE_HEADERS` - these get processed by the log scripts, and inserted below the commit mail if present and non-empty in the commit message. You can probably remove the `PR` and `MFC after` entries. And of course you can add your own.
   - `$MAIL_BRANCH_HDR` - if you want to insert a header into each commit mail describing the branch on which the commit was made, define this to match your setup. Or leave it empty if you do not want such a header.
   - `@COMMIT_HOSTS` - define this to be a list of hosts on which people can commit.
   - `$MAILADDRS` - set this to the admin or list address that should receive commit mail.
   - `@LOG_FILE_MAP` - change this array as you wish - each regexp is matched on the directory of the commit, and the commit log message gets stored in the `commitlogs` subdirectory in the filename mentioned.
   - `$COMMITCHECK_EXTRA` - if you do not want to use the FreeBSD specific access checks, you should remove the definition of `$COMMITCHECK_EXTRA` from this file.

     Note: Changing the `$IDHEADER` parameter is only guaranteed to work on FreeBSD platforms; it depends on FreeBSD specific modifications to **CVS**.

   You can check `cfg.pm` to see which other options can be changed, but the above is a reasonable subset.

4. Edit `exclude` to remove the FreeBSD specific entries (such as all lines beginning with `^ports/` etc.). Furthermore, comment out the lines beginning with `^CVSROOT/`, and add one line with only `^CVSROOT/` on it. After the wrapper is installed, you can add your header to the files in the `CVSROOT` directory and restore these lines, but for now they will only be in the way when you try to commit later on.
5. Edit `modules`, and delete all FreeBSD stuff. Add your own modules if you wish.
6.

Note: This step is only necessary if you specified a value for `$IDHEADER` in `cfg_local.pm` (which only works using a FreeBSD modified **CVS**).

Edit `options` to match the tag you specified in `cfg_local.pm`. A global search and replace of `FreeBSD` with your tag should suffice.

7. Edit `rcstemplate` to contain the same keywords as specified in `cfg_local.pm`.
8. Optionally remove the FreeBSD checks from `tagcheck`. You can simply add `exit 0` to the top of the file to disable all checks on tagging.
9. The last thing to do before you are finished, is to make sure the commitlogs can be stored. By default these are stored in the repository, in the `commitlogs` subdirectory of the `CVSROOT` directory. This directory needs to be created, so do the following:

```
% mkdir commitlogs
% cvs add commitlogs
```

Now, after careful review, you should commit your changes. Be sure that you have granted yourself access to the `CVSROOT` directory in your `avail` before you do this, because otherwise you will lock yourself out. So make sure everything is as you intend, and then do the following:

```
% cvs commit -m '- Initial FreeBSD scripts commit'
```

2.7 Testing the setup

You are ready for the first test: a forced commit to the `avail` file, to make sure everything works as expected.

```
% cvs commit -f -m 'Forced commit to test the new CVSROOT scripts' avail
```

If everything works, congratulations! You now have a working setup of the FreeBSD scripts for your repository. If **CVS** still complains about something, go back and recheck if all of the above steps have been performed correctly.

3 FreeBSD specific setup

The FreeBSD project itself uses a slightly different setup, which also uses files from the `freebsd` subdirectory of the FreeBSD `CVSROOT`. The project uses this because of the large number of committers, which all would have to be in the same group. So, a simple wrapper was written which ensures that people have the correct credentials to commit, and then sets the group id to that of the repository.

If your repository also needs this, the steps to set this up are documented below. But first an overview of the files involved.

3.1 Files used in the FreeBSD setup

- `access` - this file controls access information. You should edit this file to include all members of your project.

- `freebsd/commitmail.pl` - this file is not used any more, but kept for historic reasons. You should not have to touch this file.
- `freebsd/cvswrap.c` - this is the source to the CVS wrapper that you will need to install to make all access checks actually work. More information on this below. You should edit the paths in the `ACCESS` and `REALCVS` macros to match your setup.
- `freebsd/mailsend.c` - this file is needed by the FreeBSD setup of the mailing lists. You should not have to touch this file.

3.2 The procedure

1. Edit the `access` file to contain only your username.
2. Edit `cvswrap.c` to contain the correct path for your setup. This is defined in a macro named `ACCESS`. You should also change the location of the real `cvs` binary if it is not appropriate to your situation. The stock `cvswrap.c` expects to be a replacement for the systemwide cvs command, which will be moved to `/usr/bin/ncvs`.

   My copy of `cvswrap.c` has this:

   ```
   #define ACCESS "/local/cvsroot/CVSROOT/access"
   #define REALCVS "/usr/bin/ncvs"
   ```

3. Next up is installing the wrapper to ensure you become the correct group when committing. The sources for this live in `cvswrap.c` in your `CVSROOT`.

   Compile the sources that you edited to include the correct paths:

   ```
   % cc -o cvs cvswrap.c
   ```

   And then install them (you have to be root for this step):

   ```
   # mv /usr/bin/cvs /usr/bin/ncvs
   # mv cvs /usr/bin/cvs
   # chown root:ncvs /usr/bin/cvs /usr/bin/ncvs
   # chmod o-rx /usr/bin/ncvs
   # chmod u-w,g+s /usr/bin/cvs
   ```

   This installs the wrapper as the default `cvs` command, making sure that anyone who wants to use the repository has to have the correct access levels.
4. You can now remove everyone from your repository group. All access control is done by your wrapper, and this wrapper will set the correct group for access.

3.3 Testing the setup

Your wrapper should now be setup. You can of course test this by making a forced commit to the `access` file:

```
% cvs commit -f -m 'Forced commit to test the new CVSROOT scripts' access
```

Again, if this fails, check to see whether all of the above steps have been executed correctly.