Howto: CVS to Git

Efficient Multi-Module CVS Migration

Simon Cook
Embecosm

Application Note 11. Issue 1
Publication date February 2013

Legal Notice

Table of Contents

1. Introduction 1
    1.1. Target Audience 1
    1.2. Example 1
    1.3. Further information 1
    1.4. About Embecosm Application Notes 2
2. Mirroring a *CVS* repository 3
3. Splitting up the Repository 4
4. Conversion to Git 6
5. Building a Script 8
6. Summary 15

Glossary 16

References 17

List of Figures

3.1. **synccvs** shell function 5
5.1. Complete *CVS* to *Git* conversion script 10

Chapter 1. Introduction

For many years, *CVS* has been the primary tool used by teams of developers on large software code bases, allowing each developer to work on a single master version of the software. This tool is in continued use today, despite the introduction of alternatives which do not suffer from many of the disadvantages found in *CVS*. One such popular example is *Git*, a distributed versioning tool created and popularized by Linus Torvalds.

Tools exist for migrating repositories from *CVS* to *Git*, but these can suffer from issues such as being buggy and slow when used with remote repositories, issues with repeating path names and non-maintainability (once a repository has been converted, it cannot be fast-forwarded to the latest version). The issue of multiple modules becomes more complex when there are modules whose name is also a top-level directory in the source tree, but where the module does not consist of just that directory; some tools check out just the directory and others the module.

This application note covers the process of migrating multiple *CVS* modules from a remote repository to separate *Git* repositories with these tools in a reliable way, whilst minimizing network traffic to remote *CVS* servers when extracting and converting multiple modules (i.e. each file is only checked out once using this process once regardless of the number of modules that will be converted). It also solves the module-directory issue by stripping down the repository to just the required files and then cloning the new repository in its entirety.

1.1. Target Audience

This Application Note is aimed primarily at *Git* users who wish to import *CVS* repositories into their existing work flows and for developers/administrators who wish to do a permanent or continuous one-way migration from *CVS* to *Git*. This note assumes basic familiarity with *Git* (i.e. the concept of a remote and a repository) but does not assume familiarity with *CVS*. This script also assumes understanding of a user's preferred shell; commands written are compatible with the *bash* shell but should easily be convertible to another shell.

1.2. Example

An example of this system is Embecosm's mirror of the *sourceware CVS* repository, mirroring the *CGEN* module as a *Git* repository. This can be found at http://github.com/embecosm/cgen and is actively maintained via the same process described here. This mirror is used as an example throughout this application note.

For the code examples used in this application note, the following shell variables should be set as follows.

```
${DESTDIR} : destination directory for module specific CVS repo
${GITDIR}  : destination directory for module specific Git repo
${REPOURL} : URL to push the module specific Git repository to
${SRCDIR}  : source directory for storing initial local copy of CVS repo
```

1.3. Further information

The main source of information regarding the use of *Git* is the *Git* documentation [1]. This documentation can either be browsed on the *Git* website or is installed as man pages with the main *Git* distribution.

Similarly, documentation for *CVS* [2], is also included with its distribution.

1.4. About Embecosm Application Notes

Embecosm publishes a series of free and open source application notes, designed to help working engineers with practical problems.

Feedback is always welcome, which should be sent to <info@embecosm.com>.

Chapter 2. Mirroring a CVS repository

Note
This stage is only needed for remote repositories. When using local repositories, the local source should instead be used for further steps.

The first step in converting to *Git* is to replicate the *CVS* repository so there a local copy is available. This is done so that when splitting up the repository into those for each module, files and directories that are common to multiple repositories are only downloaded once. In addition, any directory naming issues that arise can simply be solved by renaming the local directory. Once complete, this can be split up to create new *CVS* and *Git* repositories as required.

There are two ways in which this can be done. The first method uses *rsync* (or other similar tool) to clone the repository, if such access to the bare repository available. Using *rsync* for this task has the benefit that it minimizes network traffic when updating this copy in the future and additionally allows directories to be omitted as required, for example those that will not feature in any converted repository.

For this purpose, the **CVSROOT** directory can be omitted. In the authors experience, the directory is approximately half the size of the entire repository and is not used in any other conversion step, so not downloading this directory dramatically reduces the time and resources needed to complete the conversion.

```
rsync -az -v --delete --delete-excluded --exclude CVSROOT/** \
  sourceware.org::src-cvs/ ${SRCDIR}
```

The second method, which can be used if bare repository access is not possible is to use a tool like **cvsclone** to duplicate the entire repository (using the module name "**.**" or the name of the repositories top level directory).

Note
cvsclone is not always found in an operating system's software repository. For this guide, the author obtained a working copy of **cvsclone** from https://github.com/akavel/cvsclone.

For example, to clone the *sourceware* repository, the following command would be used. The tool accepts two parameters, the first being the location of the *CVS* repository to clone (after the **-d** argument) and the second being the module/directory to be cloned.

```
cvsclone -d :pserver:anoncvs@sourceware.org:/cvs/src src
```

The **-d** parameter specifies the *CVS* server to connect to, with **src** being the module to clone.

Note
cvsclone takes a lot of time to complete creating a new clone. This is due to it checking out each version of a file sequentially. For large repositories it could take several days for the initial clone operation to complete.

Chapter 3. Splitting up the Repository

CVS repositories have a concept of modules, which are different views into a single repository. As an example repository, module *a* could contain files **foo** and **bar** whilst module *b* contains **bar** and **baz**. When a change to **bar** is made from the perspective of either module, the same change appears in the other as they are in the same repository and share the same **bar**.

Since *Git* has no equivalent concept, we will create a separate *Git* repository for each *CVS* module. The first step in this is to split the local *CVS* repository into multiple repositories, one for each module. These will contain just the directories and files which make up each module. *rsync* is again used to copy these files from the master *CVS* repository to where the modules repository is stored.

Note

This will result in duplication of files that are common to multiple modules. Using the example above, **bar** will be copied to the repository for both the *a* and *b* modules.

Note

Although this section specifically refers to cloning individual *CVS* modules, it is not limited to this. Any arbitrary selection of files will work in the same way and achieve the same result.

To aid in this, a shell function **synccvs** has been created which copies files out of the correct locations so that the complete repository history is preserved.

In *CVS*, tracked files are stored in **,v** files, which contains each file's state across all branches and tags, including their commit metadata. If the file is later deleted, it is moved to a directory called **Attic**. **synccvs** checks this location in addition to the **HEAD** repository directory to ensure history is preserved wherever it is stored.

The **synccvs** shell function is listed in Figure 3.1. It accepts one argument, the name of the file or directory to be stored. It then copies the correct files from the master repository (defined as **${SRCDIR}**) to the module specific directory (**${DESTDIR}**).

```
# Function for syncing file out of repository
synccvs() {
  # Make sure parent directory works, otherwise rsync fails
  mkdir -p `dirname ${DESTDIR}/${1}`
  # Firstly if directory, rsync dir
  if test -d ${SRCDIR}/${1}; then
    rsync -avz ${SRCDIR}/${1}/ ${DESTDIR}/${1}
    return
  fi
  # Next, if file not in attic, rsync that
  if test -e ${SRCDIR}/${1},v; then
    rsync -avz ${SRCDIR}/${1},v ${DESTDIR}/${1},v
    return
  fi
  # Finally, check if file in attic, then rsync that
  if test -e `dirname ${SRCDIR}/${1}`/Attic/`basename ${SRCDIR}/${1}`,v; then
    mkdir -p `dirname ${DESTDIR}/${1}`/Attic
    rsync -avz `dirname ${SRCDIR}/${1}`/Attic/`basename ${SRCDIR}/${1}`,v \
      `dirname ${DESTDIR}/${1}`/Attic/`basename ${DESTDIR}/${1}`,v
    return
  fi
  echo "Path doesn't exist! ${1}"
  exit 1
}
```

Figure 3.1. synccvs shell function

In the following example, this function is used to copy a project's change log (stored in a file called **ChangeLog**) to the new repository. In the case of the sourceware repository, this would be done via the following command.

```
synccvs src/ChangeLog
```

Finally, once **synccvs** has been called for all files/folders to be included, the *CVS* repository can be reinitialized, creating a **CVSROOT** directory for the new repository.

```
cvs -d ${DESTDIR} init
```

Chapter 4. Conversion to Git

With separate *CVS* repositories created for each module for conversion, the final stage is to convert these to *Git* using the **git cvsimport** command, adding remote destinations for each repository and finally pushing these to their remotes.

git cvsimport uses a tool called **cvsps** to create patch sets which are then imported into the *Git* repository. **cvsps** keeps a cache of what has previously been imported into the repository. There is a risk of this cache causing the repository to update incorrectly during successive updates to the *Git* repository due to the *CVS* reinitialization. This cache should therefore be removed in order to ensure the repository is correctly maintained.

The following two line script uses the location of the *CVS* repository and deletes its corresponding **cvsps** cache file.

```
CVSPSFILE=`echo ${DESTDIR} | sed 's/\//\#/g'`
rm -Rf ~/.cvsps/${CVSPSFILE}*
```

Note

There are many different versions of **cvsps**, each of which have different issues importing different repositories. The author has had the best experience with the following version:

https://github.com/Alexpux/cvsps

The *Git* repository is then created using **git cvsimport**, with the source and destination directories set as required. For the entire repository to be imported as-is, **.** can again be used as the module name. Alternatively the name of a subdirectory can be used, as in the example below. With keeping with the standard *Git* naming conventions, the **HEAD** branch has been named **master**. The parameters specified when using this program are specified below.

Note

If the *CVS* repository already has a branch named **master**, the name of the **HEAD** branch should be set to something other than **master** in the following command, otherwise the import will not occur correctly.

```
git cvsimport -d ${DESTDIR} -C ${GITDIR} -p -z,120 -o master -k src
```

-d specifies the *CVS* repository, which in this case is the one previously created when splitting up the main repository.

-C does the same for the respective *Git* repository.

-p -z,120 specifies the amount of time in seconds that two file commits can differ in *CVS* and still be classed as the same commit under *Git*. The default if this option is omitted is 300 seconds (5 minutes). A value of 2 minutes is used here for demonstration purposes.

-o specifies the name to set the **MAIN** branch to have in *Git* (usually **master**).

-k sets the directory to treat as the root of the repository. Usually this will be **.** but in this example, due to the structure of the particular *CVS* repository (all source code is stored in a directory called **src**), this is used instead.

Once complete, a remote can be added to the repository and it can be pushed with the **--mirror** option set, forcing all branches and tags to be in sync with the original *CVS* repository.

```
git remote add origin ${REPOURL}
git push origin --mirror
```

Chapter 5. Building a Script

With the steps in the previous chapters completed, a clone of the *CVS* repository has been made and a *Git* repository made of a module. It may however be desired to do this for many modules and to periodically update these repositories (if for example work is still carried out in *CVS*). As such, creating a script to extend and automate this process is desirable.

This script should consist of a step to clone the repository (if needed), followed by sections to handle each module to be converted. The script in Section 5.1 is used to maintain Embecosm's *CGEN* repository and demonstrates the form such a script could take. It is also available under a GPL v3 license and forms Embecosm Software Package 8 (ESP 8), available at http://www.embecosm.com/resources/software.

Note

For clarity, the script has had its **synccvs** function collapsed, but is identical to the one used above.

In this example, each module is enabled and disabled in the top configuration section and has its destination set, followed by the initial *CVS* cloning operation.

```
# CGEN
CGEN=1
CGENREPO="git@github.com:embecosm/cgen.git"
# Get sources (we don't check out CVSROOT because we don't use it)
export SRCDIR=${BASEDIR}/sourceware
rsync -az -v --delete --delete-excluded --exclude CVSROOT/** \
  sourceware.org::src-cvs/ ${SRCDIR}
```

Next, each module in turn has its directories set and relevant files are copied via **synccvs**.

For ease of maintenance, where the same multiple files or directories need importing into a number of repositories, the task of copying these should be placed into a separate function and that called instead. The following example is taken from converting the *sourceware* repository of GNU tool chain components. This repository contains modules for various tools (binutils, GDB, CGEN, etc.) **syncsrcsupport** copies the support files (e.g. **configure**) that are common to all compilable modules within the *sourceware* tree. (These files are contained within the **src-support** module in the base *CVS* repository.)

Note

The list of *CVS* modules along with the directories and files contained within each module can be found in the **CVSROOT/modules** file in the remote repository. This is not the same as the regenerated **CVSROOT/modules** files found in each modules repository. This file can either be downloaded separately via **rsync**, **cvs** or though a **cvsweb** interface if one is available.

```
# This function acts as an alias for synccvsing the src-support module found in
# CVSROOT/modules on sourceware
syncsrcsupport() {
  synccvs src/.cvsignore
  synccvs src/COPYING
  synccvs src/COPYING3
  synccvs src/COPYING.LIB
  synccvs src/COPYING3.LIB
  synccvs src/COPYING.NEWLIB
  synccvs src/COPYING.LIBGLOSS
  synccvs src/ChangeLog
  synccvs src/MAINTAINERS
  synccvs src/Makefile.def
  ... other files not listed ...
}
```

Once all files have been synchronized, the *Git* repository is then created and uploaded as previously described.

```
  # Reinitialize cvs for our new repo and then convert (using src as module)
  cvs -d ${DESTDIR} init
  git cvsimport -v -d ${DESTDIR} -C ${GITDIR} -p -z,120 -o master -k src
  # Push to GitHub
  cd ${GITDIR}
  git remote rm github || true
  git remote add github ${CGENREPO}
  git push github --mirror
```

Note

It is worth nothing in this example that the original *Git* remote **github** is first removed before being set. This enables the destination to be changed via the configuration. This command is also combined with **true** to calculate its return value. This is because the script is set to quit on the first error and attempting to remove a non-existent remote would cause an error.

If the remote does not exist (or it cannot be written to) then an error is caught during the **git push** command and the script exits as intended.

Adding other modules to the script is done by duplicating the section for that module, setting the files to be imported and making the appropriate changes to the configuration.

Finally, this script can be added to a crontab to allow the repositories to be automatically updated periodically. Alternatively, this script could be added as part of the post commit filters (set by changing the **CVSROOT/loginfo** file), enabling the *Git* repository to be updated as soon as a new *CVS* commit is made. Examples of both of these are shown below.

```
# Update CVS mirrors
0 * * * * /path/to/mirror/script/mirrorscript
```

```
* /path/to/mirror/script/mirrorscript %s
```

```
#!/bin/bash -e
# A script to convert sourceware's CVS repo to a set of Git repos
# Written by Simon Cook <simon.cook@embecosm.com>

# Copyright (c) 2013 Embecosm Limited

# This program is free software; you can redistribute it and/or modify it
# under the terms of the GNU General Public License as published by the Free
# Software Foundation; either version 3 of the License, or (at your option)
# any later version.

# This program is distributed in the hope that it will be useful, but WITHOUT
# ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
# FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License for
# more details.

# You should have received a copy of the GNU General Public License along
# with this program.  If not, see <http://www.gnu.org/licenses/>.

########################################
##           Configuration            ##
########################################

# 1. Location to store working directory (script will only work in there, except
#    for removing CVSPS working files on each iteration)
BASEDIR='/opt/sourcewaretree'
# 2. Configuration for which repositories to upload
#    To enable sync and upload, set the enable variable to 1 and destination
#    (I have only included a selection here, but really can do all if we want)
# CGEN
CGEN=1
CGENREPO="git@github.com:embecosm/cgen.git"
# binutils
BINUTILS=0
BINURILSREPO="git@github.com:embecosm/binutils.git"
# src - the entire tree
ALLSRC=0
ALLSRCREPO="git@github.com:embecosm/sourceware.git"

# We need a custom function to merge in changes from the different
# locations that changes may be found in.
synccvs() {
  # Make sure parent directory works, otherwise sync fails
  mkdir -p `dirname ${DESTDIR}/${1}`
  # Firstly if directory, sync dir
  if test -d ${SRCDIR}/${1}; then
    rsync -az ${SRCDIR}/${1}/ ${DESTDIR}/${1}
    return
  fi
```

Figure 5.1. Complete *CVS* to *Git* conversion script


```
  # Next, if file not in attic, sync that
  if test -e ${SRCDIR}/${1},v; then
    rsync -az ${SRCDIR}/${1},v ${DESTDIR}/${1},v
    return
  fi
  # Finally, check if file in attic, then sync that
  if test -e `dirname ${SRCDIR}/${1}`/Attic/`basename ${SRCDIR}/${1}`,v; then
    mkdir -p `dirname ${DESTDIR}/${1}`/Attic
    rsync -az `dirname ${SRCDIR}/${1}`/Attic/`basename ${SRCDIR}/${1}`,v \
      `dirname ${DESTDIR}/${1}`/Attic/`basename ${DESTDIR}/${1}`,v
    return
  fi
  echo "Path doesnt exist! ${1}"
  exit 1
}

# This function acts as an alias for synccvsing the src-support module found in
# CVSROOT/modules on sourceware
syncsrcsupport() {
  synccvs src/.cvsignore
  synccvs src/COPYING
  synccvs src/COPYING3
  synccvs src/COPYING.LIB
  synccvs src/COPYING3.LIB
  synccvs src/COPYING.NEWLIB
  synccvs src/COPYING.LIBGLOSS
  synccvs src/ChangeLog
  synccvs src/MAINTAINERS
  synccvs src/Makefile.def
  synccvs src/Makefile.in
  synccvs src/Makefile.tpl
  synccvs src/README
  synccvs src/README-maintainer-mode
  synccvs src/compile
  synccvs src/config
  synccvs src/config-ml.in
  synccvs src/config.guess
  synccvs src/config.if
  synccvs src/config.rpath
  synccvs src/config.sub
  synccvs src/configure
  synccvs src/configure.ac
  synccvs src/configure.in
  synccvs src/contrib
  synccvs src/depcomp
  synccvs src/etc
  synccvs src/gettext.m4
  synccvs src/install-sh
  synccvs src/lt~obsolete.m4
```

```
  synccvs src/ltgcc.m4
  synccvs src/ltsugar.m4
  synccvs src/ltversion.m4
  synccvs src/ltoptions.m4
  synccvs src/libtool.m4
  synccvs src/ltcf-c.sh
  synccvs src/ltcf-cxx.sh
  synccvs src/ltcf-gcj.sh
  synccvs src/ltconfig
  synccvs src/ltmain.sh
  synccvs src/makefile.vms
  synccvs src/missing
  synccvs src/mkdep
  synccvs src/mkinstalldirs
  synccvs src/move-if-change
  synccvs src/setup.com
  synccvs src/src-release
  synccvs src/symlink-tree
  synccvs src/ylwrap
}

# Get sources (we don't check out CVSROOT because we don't use it)
export SRCDIR=${BASEDIR}/sourceware
rsync -az -v --delete --delete-excluded --exclude CVSROOT/** \
  sourceware.org::src-cvs/ ${SRCDIR}

########################################
##            cgen Module             ##
########################################
if test ${CGEN} == 1; then
  export DESTDIR=${BASEDIR}/cgen
  export GITDIR=${BASEDIR}/cgen.git
  # Sync CVS Tree
  rm -Rf ${DESTDIR}
  mkdir -p ${DESTDIR}
  syncsrcsupport
  synccvs src/CVS
  synccvs src/cgen
  synccvs src/cpu
  # Remove cvsps temporary files
  CVSPSFILE=`echo ${DESTDIR} | sed 's/\//\#/g'`
  rm -Rf ~/.cvsps/${CVSPSFILE}*
  # Reinitialize cvs for our new repo and then convert (using src as module)
  cvs -d ${DESTDIR} init
  git cvsimport -v -d ${DESTDIR} -C ${GITDIR} -p -z,120 -o master -k src
  # Push to GitHub
  cd ${GITDIR}
  git remote rm github || true
  git remote add github ${CGENREPO}
```

```
  git push github --mirror
fi

########################################
##          binutils Module           ##
########################################
if test ${BINUTILS} == 1; then
  export DESTDIR=${BASEDIR}/binutils
  export GITDIR=${BASEDIR}/binutils.git
  # Sync CVS Tree
  rm -Rf ${DESTDIR}
  mkdir -p ${DESTDIR}
  syncsrcsupport
  synccvs src/CVS
  synccvs src/binutils
  synccvs src/opcodes
  synccvs src/bfd
  synccvs src/libiberty
  synccvs src/include
  synccvs src/gas
  synccvs src/gprof
  synccvs src/ld
  synccvs src/gold
  synccvs src/elfcpp
 synccvs src/intl
  synccvs src/texinfo
  synccvs src/cpu
  # Remove cvsps temporary files
  CVSPSFILE=`echo ${DESTDIR} | sed 's/\//\#/g'`
  rm -Rf ~/.cvsps/${CVSPSFILE}*
  # Reinitialize cvs for our new repo and then convert (using src as module)
  cvs -d ${DESTDIR} init
  git cvsimport -v -d ${DESTDIR} -C ${GITDIR} -p -z,120 -o master -k src
  # Push to GitHub
  cd ${GITDIR}
  git remote rm github || true
  git remote add github ${BINUTILSREPO}
  git push github --mirror
fi

########################################
##      src Module (everything)       ##
########################################
if test ${ALLSRC} == 1; then
  export DESTDIR=${BASEDIR}/allsrc
  export GITDIR=${BASEDIR}/allsrc.git
  # Sync CVS Tree
  rm -Rf ${DESTDIR}
  mkdir -p ${DESTDIR}
```

```
  synccvs src
  # Remove cvsps temporary files
  CVSPSFILE=`echo ${DESTDIR} | sed 's/\//\#/g'`
  rm -Rf ~/.cvsps/${CVSPSFILE}*
  # Reinitialize cvs for our new repo and then convert (using src as module)
  cvs -d ${DESTDIR} init
  git cvsimport -v -d ${DESTDIR} -C ${GITDIR} -p -z,120 -o master -k src
  # Push to GitHub
  cd ${GITDIR}
  git remote rm github || true
  git remote add github ${ALLSRCREPO}
  git push github --mirror
fi
```

Chapter 6. Summary

The following list can be used as a summary for porting a *CVS* repository to a set of *Git* repositories. It serves as a list of functionality that any updating script should have.

1. Clone the repository (if needed) so that there is a local copy of the *CVS* repository to work from (Chapter 2).
2. Split up the repository copy into separate repositories for each module, each of which will become *Git* repositories, utilising the **synccvs** function as required (Chapter 3).
3. Convert each component to a *Git* repository using **cvsps** and **git cvs-import** and push these repositories to their respective remotes (Chapter 4).
4. (Optional) Set up a cron job or *CVS* post commit filter to automatically incorporate changes (Chapter 5).

Glossary

CVS
: (Concurrent Versions System), open source client-server revision control/source code management system.

Git
: Open source distributed revision control/source code management system originally developed by Linus Torvalds for the Linux kernel.

Repository
: A store for source code (and other files) which maintains records of which files were modified at a particular time, what those changes were and who made them.

References

[1] *Git* Documentation Available at http://git-scm.com/doc.

[2] *CVS* Documentation Available at http://ximbiot.com/cvs/manual.

