Interchange + CVS HOWTO

Table of Contents

Table of Contents

1. Introduction

1.1. Preamble

1.2. Purpose

The purpose of this document is to help others take advantage of CVS and Interchange together to increase the quality of their programming, whether they are sole developers or part of a large team of programmers, graphic artists, and HTML design gurus. Portions of it apply to general CVS setup and use, but it is geared toward the average developer using Interchange to implement an e−commerce website.

1.3. Audience

I intend for this document to be useful to those who are not yet familiar with CVS as well as those who are. If you already know how to setup a pserver then you might just skim chapter 2 ("Setup CVS"), or skip it all together.

In addition, I have tried to write at a technical level that would be on par with what I perceive to be the average Interchange user that participates on the interchange−users mailing list. It is assumed that the reader can and already has setup Interchange and the template catalog (such as foundation or construct) is working correctly.

1.4. Contact the author

If you find any spelling errors, technical slip−ups, mistakes, subliminal messages, or if you wish to send feedback, critique, remarks, comments, or if you wish to contribute examples, instructions for alternative platforms, chapters, or other material, please do so.

The preferred method of submitting changes is in the form of a context diff against the SDF source file (ic_cvs.sdf). Please address your correspondence to:

Dan Browning danpb@mail.com

1.5. The advantages of using CVS

CVS is a very useful tool and can help you in your development, no matter if you are one developer or are part of a team of developers.

- What is CVS all about?
- What are its advantages?

The official CVS website (http://www.cvshome.org/new_users.html) has more detailed answers to these questions, but here are some brief points of interest.

- Checkout "historic" points in time or milestones in a project, for example when an e−commerce site went "live" or before a major branch in the code.
- Revert to older versions of a file, directory, or an entire website.
- Branching releases. Concurrently develop an unstable development version as well as fix bugs in the stable production version.
- Multiple developers can work on the same catalog and even the same file at the same time. (For more information about how multiple simultaneous writes are merged and conflicts resolved, see the CVS docs in the Resources Appendix).
- CVS is better than ftp for file transfer, because it automatically downloads only changed files, and even then, only the portion of the file that has changed (using patches).
- CVS can automatically merge two simultaneous writes to the same file by different developers.
- Allows one to keep track of the changes that have been made over time (many release managers repackage CVS commit logs into WHATSNEW, HISTORY, and/or NEWS files).

1.6. How to use this document

There are many potential uses of CVS as it applies to Interchange. In fact, there are as many unique ways to use CVS as there are unique developers. This document only covers some of the ways, including basic and useful techniques to get started using CVS. For the intents of the average web developer using IC for a B2C e−commerce site, I've identified a few of the possible uses:

Simple

- One server
- One catalog
- One CVS module
- One branch

Medium

- One server
- Two catalogs (e.g., one is live, one is development)
- Two CVS modules
- Separate development and live branches

Complex/Custom

- Multiple servers (e.g., developers' servers, staging servers, and live servers)
- Multiple catalogs
- Multiple CVS modules
- Multiple branches
- Custom setup

This document attempts to cover the simple well, and explain many aspects of the medium. Which will hopefully give you the background you need if you decide to setup your own complex development environment.

2. Setup CVS

2.1. Assumptions

Here are some of the assumptions that I make that apply to various parts of the rest of this document:

- Red Hat Linux 7.x
- Interchange installed (RPM or tarball)
- Default Interchange tarball installation directory paths (adjust for your environment)
- Template catalog setup and working

Note: I will assume "foundation" for the catalog name and directory paths, but it should be just as easy to use this document with the construct catalog or your own catalog by mentally transposing the names and paths.

There shouldn't be any reason why you could not do everything I mention here on other Linux distributions, Unices or Windows (using cygwin). However, my statements will reflect Red Hat Linux 7.x. Additionally, Red Hat Linux 6.x is for the most part the same as 7.x, except for the difference of using inetd instead of xinetd to setup pserver.

2.2. Install CVS

This is the easy part. For Red Hat Linux systems, download the CVS rpms and install them. The following RPM command will download and install the Red Hat 7.1 version of CVS from rpmfind.net.

Note: You need to be root to complete the following tasks

```
su − root
rpm −Uvh ftp://speakeasy.rpmfind.net/linux/redhat/7.1/en/os/i386/RedHat/RPMS/cvs−1.11−3.i386.rpm
```

Create the user and group that will administrate the Interchange repository. For this document, it will be the interch user, (which was setup during the installation of Interchange). But if you understand the mechanics of Unix users/groups, then you can use whatever username and group scheme you prefer. For example, some create a cvs user and cvs group, then add the Interchange user and catalog owner to its group or vice−versa. The integration of Interchange and CVS in the latter portion of this document will require that the CVS user can write to the catalog directory.

2.3. Create the CVS repository directory

You will need to create a repository directory such as `/rep`, which is used here and in the rest of the document, but it can be any directory you desire, and must be owned by the cvs user. Many use `/var/rep` or `/home/cvs/rep`.

```
su − root
mkdir /rep
chown interch.interch /rep
```

2.4. Setup environment variables

The CVSROOT and EDITOR environment variables should be setup for all users in /etc/profile. Of course, EDITOR can be whatever Unix text editor you prefer, such as `vi`, `emacs`, `pico`, or `joe`.

`/etc/profile:`

```
export CVSROOT=/rep
export EDITOR=vi
```

Note: You will need to logout/login for the profile changes to take effect.

2.5. Initialize the repository

Initialize the repository as the CVS user, which is `interch` for this document.

```
su - interch
cvs -d /rep init
```

2.6. CVS Authentication

2.6.1. Background

Authentication is done in CVS through the `$CVSROOT/CVSROOT/passwd` file. It can be easily manipulated through some of the CVS administration tools that are available.

2.6.2. CVS administration tools

- http://freshmeat.net/projects/cvsadmin/
- http://freshmeat.net/projects/cvspadm/

I recommend cvsadmin, but there are also a variety of manual methods that can be used in the absence of such tools, one of which involves copying the system shadow file and modifying it for use by CVS. For more information on this manual method, see the Red Hat CVS pserver setup guide by Michael Amorose (http://www.michael-amorose.com/cvs/).

2.6.3. Setup authentication using the cvsadmin tool

You can find a tarball to install on your system using the above address, but here is the address of a recent RPM package of the version. This package is intended for Mandrake systems, but is compatible with Red Hat Linux 7.1:

- ftp://speakeasy.rpmfind.net/linux/Mandrake-devel/contrib/RPMS/cvsadmin-1.0.1-1mdk.i586.rpm

After installing, create a password file (*touch $CVSROOT/CVSROOT/passwd*), and execute `cvsadmin add <usernames>`.

2.7. Setup CVS modules

Note: From this point on, assume that all commands are executed as the CVS user (e.g. interch), unless otherwise specified.

A module is CVS is like the concept of a "project", where each module has its own branches, trees, and other features.

2.7.1. Add your project to the `modules` configuration file

The format of the modules file is explained in detail in the CVS documentation, here is the simplest way to use it:

`/rep/CVSROOT/modules:`

```
<Module name><TAB><Module Directory>
```

The module name can be whatever you want, and the module directory is what we will create later under /rep. We'll want a module for the template catalog (foundation). For example:

```
foundation      foundation
```

2.7.2. Create the module directory

This is the directory that is referred to in the `CVSROOT/modules` file we just modified.

```
mkdir /rep/foundation
```

2.8. Setup binary file types

This isn't necessary if you aren't going to manage any binary files (e.g. if you plan on excluding your /images/ directory). But I recommend including it. The following is an example including many binary file types (by extension) used in web development.

`/rep/CVSROOT/cvswrappers:`

```
*.avi   -k 'b' -m 'COPY'
*.doc   -k 'b' -m 'COPY'
*.exe   -k 'b' -m 'COPY'
*.gif   -k 'b' -m 'COPY'
*.gz    -k 'b' -m 'COPY'
*.hqx   -k 'b' -m 'COPY'
*.jar   -k 'b' -m 'COPY'
*.jpeg  -k 'b' -m 'COPY'
*.jpg   -k 'b' -m 'COPY'
*.mov   -k 'b' -m 'COPY'
*.mpg   -k 'b' -m 'COPY'
*.pdf   -k 'b' -m 'COPY'
*.png   -k 'b' -m 'COPY'
*.ppt   -k 'b' -m 'COPY'
*.sit   -k 'b' -m 'COPY'
*.swf   -k 'b' -m 'COPY'
*.tar   -k 'b' -m 'COPY'
```

```
*.tgz   -k 'b' -m 'COPY'
*.tif   -k 'b' -m 'COPY'
*.tiff  -k 'b' -m 'COPY'
*.xbm   -k 'b' -m 'COPY'
*.xls   -k 'b' -m 'COPY'
*.zip   -k 'b' -m 'COPY'
```

2.9. Testing your repository

At this point, you should have a working (though empty) CVS repository. Before we continue with setting up the pserver or importing source code, try logging in as one of the CVS users listed in your CVSROOT/passwd and test the checkout.

```
#test checkout in home directory of any cvs user
mkdir ~/src
cd ~/src
cvs co foundation
```

This should create `foundation/` and `foundation/CVS`.

2.10. Setup the CVS pserver

You will likely need to be root to do this, and there are lots of guides on the Internet for setting up a CVS pserver, hopefully you wont have any trouble doing it on your particular operating system. See the Resources Appendix for more information.

2.10.1. Setup pserver in Red Hat Linux 7.1 using xinetd.

For Red Hat Linux 7.x, edit `/etc/xinetd.d/cvspserver` (create a new one if none exists). The following works for me, but customization may be required for your environment (see the next section below for an inetd−based system example). This also must be done as root.

```
su − root
/etc/xinetd.d/cvspserver:

# default: on
service cvspserver
{
        disable = no
        socket_type  = stream
        protocol  = tcp
        wait   = no
        user    = root
        server   = /usr/bin/cvs
        server_args  = −f −−allow−root=/rep pserver

}
```

Also unset the HOME variable in xinetd. This was required for my repository to work correctly, but if anyone has a better suggestion, I would appreciate a note.

```
/etc/xinetd.d/cvspserver:

unset HOME
```

Now, restart xinetd for the changes to take effect.

```
service xinetd restart
```

2.10.2. Setup pserver in inetd−based systems.

For inetd−based systems such as Red Hat Linux 6.2, make sure that the following files are setup accordingly.

/etc/services:

```
cvspserver      2401/tcp
```

N:**/etc/inetd.conf:**

```
cvspserver stream tcp nowait root /usr/sbin/tcpd /usr/local/bin/cvs −−allow−root=/usr/local/newre
```

2.10.3. Testing your pserver

At this point, you should be able to use a CVS client to use your pserver and execute all the same commands that you can locally (which we tested before). You may wish to take advantage of a graphical CVS client, which can be particularly helpful in leveling the learning curve.

See the Resources Appendix for links to some graphical CVS tools.

3. Import your Interchange catalog into CVS

3.1. Configuring your catalog

Eventually, we will import your catalog into the CVS repository, but first we need to do some work with a temporary copy of the catalog so we can get it into shape for importing.

Note: From here on, assume the use of the Interchange user, such as `interch`, unless otherwise noted.

```
su – interch
```

If you installed via RPM:

```
service interchange stop
```

If you installed via tarball (default path):

```
/usr/local/interchange/bin/interchange ––stop
```

3.2. Remove old CVS folders

If, for any reason, you already have `CVS/` directories in your catalog, they must be removed because they might interfere with the new CVS setup. For example, maybe you moved servers and you are setting up CVS again. You might use the following `find` command, which will find any folders named `CVS` in the current directory and remove them. There is probably a better way to deal with old `CVS/` folders, but the following works for me (again, suggestions welcome).

Note: You should make a backup of the catalog directory before you do this.

```
#Become Interchange catalog user
su – interch

#backup catalog folder first
tar czf ~/foundation_backup.tgz /var/lib/interchange/foundation

#get rid of any old CVS folders –– (BE CAREFUL!)
cd /var/lib/interchange/foundation
find . –name CVS –exec rm –Rf {} \;
```

3.3. Create a working copy of your catalog

A working copy of your catalog is necessary to get it into shape for use with CVS. The following command creates a copy in the `/tmp` directory.

```
cp –a /var/lib/interchange/foundation /tmp/import_foundation
cd /tmp/import_foundation
```

3.4. Streamline your catalog for CVS

3.4.1. Considerations about what to import into CVS

From your working directory (`/tmp/import_foundation`), decide which files will be in the CVS repository, and which will not. While it is entirely possible to import the entire catalog into the repository unchanged, I usually prefer to doctor my directories up before letting them into my repository because of several reasons:

- Will the file be modified by another source?

For example, `/etc/order.number` is modified by Interchange when run. But not everyone will use a local development model that includes running Interchange on a directly checked−out copy of their source. Which means this specific issue is avoided if you upload every edit before viewing your changes on a server.

- The likelihood that you will modify the file.

For example, if I am certain that I wont every want to modify the session/ files directly, then I probably wouldn't need to manage that through CVS, but I do import the empty session/ directory to make it easier when setting up new catalogs.

- Speed.

Managing less files in the repository takes away from the amount of time required for CVS checkout, update, branching, and other CVS actions. For most, this amount of time is small already, but it is a consideration for some.

- Ease of use.

Ease of use is one reason not to remove anything from your catalog before importing it, because it creates the ability to have a completely working catalog from just one checkout (much like the CVS tree at interchange.redhat.com). Whereas if you leave out other directories like etc/ session/ orders/, etc., then you must first combine your checkout with the other working parts of a catalog before the catalog is viable. But this is slower and will bring up lots of harmless notification and warning messages (about changed local versions) if you run Interchange on your local source copy (because Interchange will touch etc/ session/ orders/, etc. directly, and then warn that your local copy has changed from the CVS copy). You may be able to manage some of these notifications and warnings with `CVSROOT/cvsignore` or `$CVSIGNORE`, see the Resources appendix for more details.

3.4.2. Remove files that aren't needed in CVS

Here is an example of some directories to remove. If you do move more directories, be sure to move them to a directory that you can later use to re−unite with a checked−out copy for a working catalog. But here I chose just to move files that are not needed for a template "skeleton" catalog.

The images directory is typically symlinked to /var/www/html/foundation/images, so I remove this symlink from the working copy, and replace it with an exact copy which will go into the CVS repository.

```
cd /tmp/import_foundation
mkdir /tmp/import_foundation_nonCVS
```

```
#Setup images directory
rm images
cp −a /var/www/html/foundation/images .

#Remove
mv error.log logs/* orders/* session/* tmp/* upload/*  \
        /tmp/import_foundation_nonCVS
```

3.5. Import the streamlined catalog

Import the remaining portion of the catalog using the `cvs import` command, with "foundation" as the module name and repository directory name. See the CVS documentation resources mentioned in Appendix Resources for more information.

When you run the import command, it will launch $EDITOR (set to `'vi'` earlier), and ask for a message to go along with the import action. Whatever you see fit to write (e.g. "starting new CVS module with my foundation catalog...") is fine.

This example `import` command includes renaming the foundation "working" directory back to "foundation" for the import.

```
su − interch
cd /tmp/import_foundation
cvs import foundation foundation start
```

3.6. Testing the new CVS module

Now you should be able to do another test checkout or update using any CVS client, which should now download all the files that you have just imported into CVS. Additionally, you might test your newly imported code by making a change to one of your checked−out source files, saving it, then committing it.

index.html:

```
<!−−this is a test comment at the top of index.html−−>
```

Now commit the change

```
cvs commit index.html
```

Your changed version will now be resident in the repository. There are a lot of good CVS documentation and resources for discovering more about the checkout/update/commit cycle and other CVS aspects in the Resources Appendix.

You'll also notice that even if you start your interchange server, the change you made did not take effect. The next section will detail the process of tying CVS and Interchange together in a way that this will happen automatically.

4. Integrate CVS and Interchange

The next step is to allow CVS to update the directory that Interchange uses to serve pages.

4.1. CVS checkout into the catalog directory

Now it is the time to replace the directories in your catalog that have counterparts in CVS with fresh checkouts from CVS (this is a preliminary action to allow CVS to update your catalog directory when a change is made to CVS).

Note: Make sure interchange daemon is stopped and you have a good backup before continuing.

```
tar czf ~/foundation.backup2.tgz /var/lib/interchange/foundation
```

Checkout a copy from CVS into a different directory (such as `foundation_CVS`).

```
cd /var/lib/interchange/
cvs co -d foundation_CVS foundation
```

This should create the `foundation_CVS/` directory for you, so that it wont conflict with your existing `foundation/` directory.

4.1.1. Add any needed files to checked-out catalog

Note that empty directories are pruned, so they will need something in them for them to show up with a -P checkout. Often a zero-byte file called '.empty' is used.

If you removed any directories during the streamlining step, we must first add those back so that the catalog is usable to Interchange. In this document, we only removed unneeded files and left empty directories.

This can also be the time to copy any "data" files such as orders/ logs/, etc. that might be needed if it is a live catalog.

```
cd /var/lib/interchange/foundation
cp -a <NEEDED_FILES> \
        /var/lib/interchange/foundation_CVS
```

4.1.2. Install and test the new catalog

Now lets move the old `foundation` out of the way and put the new `foundation_CVS` in its place.

```
cd /var/lib/interchange/
mv foundation foundation_old
mv foundation_CVS foundation
```

Now, link up the CVS images for use by Apache.

```
cd /var/www/html/foundation/
mv images images_old
ln -s /var/lib/interchange/foundation/images images
```

Now, you should have a working catalog again. To make sure, start up Interchange and test the site with your browser.

4.2. Testing manual CVS updates on Interchange catalogs

Next, lets again update the checkout we made a while back before importing our catalog. (Alternatively, one could use a visual CVS client detailed above).

```
cd ~/src
cvs −q up −d foundation # −q for quiet, −d for directory prune/update
```

Additionally, you might test making a change to one of your checked−out source files, saving it, then committing it.

index.html:

```
<!−−this is a test comment at the top of index.html−−>
```

Now commit the change

```
cvs commit index.html
```

Your changed version will now be resident in the repository. Again, CVS documentation is in the Resources Appendix.

This time, we can allow the changes to take effect on the code being used by Interchange to serve pages. To do so, one must run a `cvs update` on the catalog directory:

```
cd /var/lib/interchange/foundation
cvs −q up −d    #up is the shortened version of "update"
```

That should notify you of the new version it downloaded with something like:

U pages/index.html

You may also get something like the following:

```
M catalog.cfg
M etc/status.foundation
M ...
? orders/000001
? ...
```

The `?` lines in the above example mean that the CVS server has never heard of the listed directories or files (they are in your local source dir but not in the CVS source dir). It is harmless, but sometimes annoying.

The `M` means that the file has been modified on your local copy, and is out of sync with the remote CVS version (e.g. when Interchange runs it updates `etc/status.foundation`). Normally this is corrected by uploading your "modified" version to the server, but in this case, the modification was done by Interchange instead of the programmer, and wasn't meant to be committed back to the CVS repository. These types of messages can be handled with `$CVSIGNORE` and `$CVSROOT/CVSROOT/cvsignore`.

Now, check to make sure that your change has taken effect by refreshing the homepage on the site. To see the

comment, use `View->Page Source` or whatever the relevant command for your browser is.

At this point, its obvious that it would be time consuming to manually run 'cvs up' every time you make a change to the source code, so the next step is to setup CVS to automatically update the catalog whenever you commit something to CVS.

4.3. Automatic updates on commit

Start by modifying $CVSROOT/CVSROOT/loginfo

```
^foundation     (date; cat; (sleep 1; cd /var/lib/interchange/foundation; cvs -q update -d) &) >>
```

The first line tells CVS that for every commit on modules that start with "foundation" (notice the regular expression `"^foundation"`), it will run `cvs update` on the given catalog directory in the background. It is important that it is executed in a forked shell (notice the `"&"`) after `sleep`'ing for 1 second, because otherwise you may run into contention issues that can cause file locking problems. The 1 second timing used above works fine for me, but a longer pause may be necessary for slower computers (you'll know if you get errors about "file locked by user"). See the CVS documentation in the Resources Appendix for more details.

4.4. Automatic e-mail on commit

Often it is very helpful to have a commit mailing list that keeps developers up-to-date on every commit happening to the CVS. To setup automatic e-mails on every commit, put the following in

/rep/CVSROOT/loginfo:

```
ALL     /usr/bin/cvs-log     $CVSROOT/CVSROOT/commitlog $USER "%{sVv}"
```

This tells CVS to pipe the commit output to a shell script, which in turn updates a log file and e-mails an update (typically to a mailing list address). Create the shell script at `/usr/bin/cvs-log` that is executable by the cvs user (using `"chmod 755 /usr/bin/cvs-log"`).

/usr/bin/cvs-log:

```
#!/bin/sh
(echo "-----------------------------------------------";
 echo -n $2"  ";
 date;
 echo;
 cat) | tee -a $1 | /usr/bin/Mail -s "[foundation-cvs] $3" foundation-cvs@example.com
```

Your commit logs will now be archived in the CVSROOT/commitlog file and e-mailed to the foundation-cvs@example.com address (which is especially useful when you have a Mailserver for CVS updates). Here is what a sample e-mail looks like:

```
Subject: [foundation-cvs] 'directory/subdirectory filename.c,1.7,1.8'


-----------------------------------------------
cvs Fri Mar 16 21:14:09 PST 2001
Update of directory/subdirectory
In directory cvs.foundationsomething.com:/tmp/cvs-serv7721
Modified Files:
filename.c
```

```
Log Message:
test
```

Now you have a working CVS development system. At this point it may be valuable to learn more about CVS the client tools that you are using.

5. The two track model: development and live catalogs

It is often very valuable to have a two−track development model that separates the classes of work into separate timing and decision categories. Some use "staging" and "production" terminology, others prefer "unstable" and "stable", "beta" and "release", or "development" and "live".

The easiest starting point for two−track development is to just use two completely separate CVS modules and catalogs. This can make a lot of sense for many situations, for example when the next revision of the site will be so different that it is for all practical purposes starting from ground zero.

A slightly more complicated solution is to use the CVS branches feature. It is more difficult to set up, but can be rewarding when used correctly.

5.1. When to branch

The first decision is when to branch the source code. For websites, this can sometimes be an easy decision like "first went live", or "site−wide overhaul", etc.

5.2. Which way to branch

There are many different ways to branch source code. What seems to be the most common method is to use the "trunk", which is the HEAD tag to CVS as the development version, and then make a branch when a stable release is to be made.

That model doesn't fit my development style at the current time, so I use the HEAD default branch as my stable live version, and use other tags (like DEV1 and DEV_REALLY_UNSTABLE) for my development branch.

You may find that you are merging (or "folding") most or all of your development branch back into your stable branch frequently. This is because unlike traditional programming where products are launched every two or three years with new features, web sites often have little fixes and new features added every day or every few weeks, with new "releases" happening more often than traditional software development (though not all web sites follow that trend). The flexibility is there to branch the source for quite some time to work on a very complex feature or complete redesign before bringing it to the live site, as well as the flexibility for day−to−day updates.

5.3. Performing the branch

To perform the branch use the `cvs tag −b <BRANCH NAME>` command. For example:

```
cvs tag −b DEV1
```

Remember that this does not change your locally checked out working directory to the new tag automatically, it only creates the branch within the CVS repository.

5.4. Setup the development catalog

Now we have a branch in CVS, but we need to tie it to something in the real world, namely, an Interchange catalog.

5.4.1. Importing the catalog

Like we did in Integrating CVS with Interchange, you must make another copy of your catalog for use as the development version. Some would like to keep the orders/, logs/, and other directories the same, but I prefer to start with a clean slate, especially since I don't plan on having any customers visit the development site. (In fact, you can restrict who can access the development URL using the Apache `<Directory> allow from...` directive).

5.4.1.1. Checkout source code

```
cd /var/lib/interchange
cvs co -d foundation_dev foundation
```

5.4.1.2. Copy any other needed directories to complete the catalog

Depending on how complete your catalog is in CVS, you may need to create or copy directories/files.

```
cd /var/lib/interchange/foundation
cp -a catalog.cfg orders/*   \
        /var/lib/interchange/foundation_dev
```

Note: A lot of the following steps are performed by the /usr/local/interchange/bin/makecat script, but here is how to do it manually:

5.4.2. Setting up a separate database

Most often, I find it profitable to make use of a second database for the development catalog, rather than having both catalogs reference the same database (especially if the first catalog is live).

5.4.2.1. Create a second database

Use the means of your database platform to create a separate database. For example, PostgreSQL users might do something like:

```
createdb foundation_dev
```

5.4.2.2. Populate the database

You can rely on the catalogs internal products/*.txt data to generate the database tables and populate them, or you can export another catalog's database and import it for the development catalog, like the example below for PostgreSQL users.

```
pg_dump foundation  > ~/foundation.dump
psql foundation_dev < ~/foundation.dump
```

5.4.3. Copy the catalog support files

```
#Must be root
su − root

#Copy HTML
cd /var/www/html/
cp −a foundation foundation_dev

#Copy CGI
cd /var/www/cgi−bin
cp −a foundation foundation_dev
```

5.4.4. Configure the Interchange daemon

Perform the necessary modifications to `interchange.cfg`. For example:

/usr/local/interchange/interchange.cfg:

```
Catalog foundation     /var/lib/interchange/foundation     /cgi−bin/foundation
Catalog foundation_dev /var/lib/interchange/foundation_dev /cgi−bin/foundation_dev
```

5.4.5. Configure the catalog specifics

The development catalog will differ at least a little bit from the standard catalog, such as in the CGI_URL and database parameters. You can also modify/add the foundation_dev/variable.txt instead of the following.

/var/lib/interchange/catalog.cfg:

```
Variable CGI_URL   /cgi−bin/foundation_dev
Variable IMAGE_DIR /foundation_dev/images
Variable SQLDSN    dbi:Pg:dbname=foundation_dev
Variable SQLDB     foundation_dev
```

Now you can restart Interchange to make your changes take effect.

5.5. Splitting updates on commit by tag

Setup CVS so that when you commit to the DEV1 branch, only the development (`foundation_dev`) catalog will be updated. And when you commit with no tags (HEAD branch), the live (`foundation`) catalog will be updated. Here is an example `loginfo`. The `−r <tag>` may be used just in case your environment is such that the tags may be changed by other sources.

$CVSROOT/CVSROOT/loginfo:

```
^foundation      (date; cat; (sleep 1; cd /var/lib/interchange/foundation_dev; cvs −q up −d; cd /
ALL     /usr/bin/cvs−log     $CVSROOT/CVSROOT/commitlog $USER "%{sVv}"
```

5.6. Using new branches

To use your new branch, checkout a working copy of the source with the correct tag specified. For example:

```
cvs co −P −r DEV1
```

Then make change to one of the files, and commit it. The change should show on your development catalog, but not your live catalog.

5.7. Merging

When you want to merge a change that you have made on your development branch into your stable branch, there are many ways that you can do it. One would be to :

- Make a change in the development branch (DEV1) and commit it.
- Copy the development−tagged file to a temporary name
- Update to the live version (HEAD)
- Overwrite the live (HEAD) version of the file with your temporary one
- Commit the result
- Update back to the development version (DEV1)

I do the above so often that I have written a Tcl script for WinCVS that will automatically perform the above steps. And similar shell scripts can probably be easily written to match your development environment.

The above seems to be the easiest way, to me. However, there are other alternatives detailed in the CVS manual in chapter 5, "Branching and merging", that I highly recommend for reading. One method involves specifying the last version that has already been merged into the live branch using a specific version number, date, relative time, or special purpose tag.

6. Tools of the trade

This is the productivity tips section, which will hopefully help you to be able to get more done in less time.

6.1. Workstation Interchange installation

Not all developers work on Linux workstations, many use Apples (graphics designers and HTML gurus tend to, I've found), and many use Windows. This means that many developers have the extra step of uploading their changes to a Unix server where Interchange is running in order to see their changes.

The remedy to that is to setup an Interchange server on your workstation, or any location that has direct access to the CVS source files. I'll explain:

The Interchange server that runs where the CVS server is (that we setup earlier) can be seen as the gathering point for all the developers. However, each developer may run as many Interchange daemons as he/she requires in a local context for the purpose of seeing the changes made before uploading them via CVS.

For example, Bob could setup another Interchange catalog on the same server as the CVS, (e.g. foundation−bob). To get direct access to those files (rather than FTP), Bob could use NFS mounts (if Bob's workstation is Linux) or SMB mounts using Samba if his workstation is a Windows variant. Any way that Bob can get direct access to the files will save him some time (by cutting out the "upload" from the "edit−>upload−>test" development cycle). One could even use VMware to run a Linux server on your Windows workstation.

Note: You can now use the cygwin compatibility confirmed in Interchange versions 4.7.6 and above to run Interchange right on your Windows workstation.

The result will be that you can modify the files with your favorite text editor and see the results immediately through your local catalog. Setting up the catalog initially is quite easy. Just follow the same steps used to setup the CVS catalog. Which is:

- Stop Interchange.
- bin/makecat a new catalog.
- Checkout from CVS into a new CVS catalog directory and link the images/ directory.
- Move any needed files back into the CVS catalog directory.
- Make modifications to products/variable.txt and catalog.cfg (e.g. CGI_URL, HOSTNAME, DBI_USER, DBI_PASSWORD).
- Restart Interchange.

One aspect of this local configuration is managing the differences between the main Interchange daemon which runs on the CVS server and the local Interchange daemon. The differences are probably hostname, database information, etc. That will all need to be managed (usually through catalog.cfg entries) and database exports & imports (i.e. the postgres pg_dump command).

Another thing that you might have noticed at this point is all the files that are modified locally by the Interchange daemon will report ? or M when you run an update. This can be handled with `CVSROOT/cvsignore` and `$CVSIGNORE`, which are beyond the scope of this document.

6.2. Mailserver for CVS updates

To setup a mailserver for CVS updates, first download and install Mailman. For RPM−based systems, check on rpmfind.net for a precompiled binary package.

After installing, read the following information about Mailman and what needs to be done after installation (taken from the RPM meta data):

"Mailman is software to help manage email discussion lists, much like Majordomo and Smartmail. Unlike most similar products, Mailman gives each mailing list a web page, and allows users to subscribe, unsubscribe, etc. over the web. Even the list manager can administer his or her list entirely from the web. Mailman also integrates most things people want to do with mailing lists, including archiving, mail <−> news gateways, and so on.

When the package has finished installing, you will need to:

- Run `/var/mailman/bin/mmsitepass` to set the Mailman administrator password.
- Edit `/var/mailman/Mailman/mm_cfg.py` to customize Mailman's configuration for your site.
- Modify the sendmail configuration to ensure that it is running and accepting connections from the outside world (to ensure that it runs, set "DAEMON=yes" in /etc/sysconfig/sendmail, ensuring that it accepts connections from the outside world may require modifying /etc/mail/sendmail.mc and regenerating sendmail.cf), and
- Add these lines:

```
ScriptAlias /mailman/ /var/mailman/cgi−bin/
Alias /pipermail/ /var/mailman/archives/public/
<Directory /var/mailman/archives>
  Options +FollowSymlinks
</Directory>
```

to `/etc/httpd/conf/httpd.conf` to configure your web server.

Users upgrading from previous releases of this package may need to move their data or adjust the configuration files to point to the locations where their data is."

Then run `/var/mailman/bin/newlist` and follow the directions from there.

6.3. Locally mapped source code for a network IC server

This is useful mostly to Windows users, since Linux users can just as easily run IC daemons on their own workstation as they can a separate server.

The idea is to have the IC server use its own files and directories for things that won't be edited and modified locally, but reference a Samba directory or NFS directory for things that will (such as `pages/`, `templates/`, etc.).

6.3.1. Mount the Samba or NFS directory

`smbmount <...>` or `mount −t nfsfs <...>`

The following script uses two directories (source and destination) to create symlinks for the commonly modified source directories of Interchange.

```
export S=/mnt/nfs/foundation
export D=/var/lib/interchange/foundation
F=db; ln -s $S/$F $D/$F
F=dbconf; ln -s $S/$F $D/$F
F=etc; ln -s $S/$F $D/$F
F=images; ln -s $S/$F $D/$F
F=pages; ln -s $S/$F $D/$F
F=special_pages; ln -s $S/$F $D/$F
F=templates; ln -s $S/$F $D/$F
```

This will leave you with a working catalog that can be quickly modified (since your editor can access the local copy), while Interchange has to do the work of going over the SMB or NFS connection.

6.4. jEdit − a good editor with Interchange/HTML/Perl colorization and CVS

I have been quite impressed with jEdit (http://www.jedit.org, and open source editor that is written in Java and runs on most platforms.

I use the interchange.xml language definition written by Chris Jesseman chris@sitemajic.net, which is available from http://www.sitemajic.net/jedit/. With this, jEdit automatically colors HTML, Perl, AND many Interchange tags very intelligently.

Further, jEdit has a CVS plugin, written by Ben Sarsgard bsarsgard@vmtllc.com, and available at: http://www.vmtllc.com/~bsarsgard/jedit.html. This plugin allows you to diff, update, and commit right from the editor.

6.5. Separate servers for development and live catalogs

If you have the luxury of separate server hardware for the development and live catalogs, you might find the following utility helpful:

- CVSviaFTP (http://www.cvshome.org/dev/addoncvsftp.html) − from the CVS Add−ons page (http://www.cvshome.org/dev/addons.html).

It allows one to have a given CVS module automatically publish each update to an FTP server, which could serve as the live server. Or one could could use it if your CVS installation is only local and you could use it to upload your changes to your production server.

A. Credits

- **Jon Jensen**: Thanks for helping me get going on the SDF format already used by the Interchange documentation, and fixing some SDF syntax errors.
- **Mike Heins & all who have contributed to the success of Interchange**: Thanks for following the Way Of The Source, for quality programming, and for helping to making IC something to write about.
- Thanks to the countless others who have written the CVS documentation that is available online, which was my only source for learning CVS.

B. Document history

- May 2001. Conceived and written by Dan Browning.
- July 19, 2001. First draft complete, first public release.
- 12 April 2002. Minor typographical edit.

C. Resources

C.1. CVS Documentation

Here are some resources for learning more about CVS. I have ranked them by the order of usefulness, which is of course, objective.

- Karl Fogel's CVS book http://cvsbook.red−bean.com/
- The official CVS manual http://www.cvshome.org/docs/manual/
- The official CVS FAQ http://faq.cvshome.org/
- The official CVS homepage http://www.cvshome.org
- Info−CVS mailing list http://mail.gnu.org/mailman/listinfo/info−cvs
- CVS FAQ 2 http://www.cs.utah.edu/dept/old/texinfo/cvs/FAQ.txt
- Sean Dreilinger's CVS Version Control for Web Site Projects http://durak.org/cvswebsites/
- Pascal Molli's CVS reference site http://www.loria.fr/~molli/cvs−index.html
- CVS Tutorial http://cellworks.washington.edu/pub/docs/cvs/tutorial/cvs_tutorial_1.html
- CVS Tutorial 2 http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/cvs/
- Red Hat CVS pserver setup guide http://www.michael−amorose.com/cvs/
- CVS Add−ons http://www.cvshome.org/dev/addons.html

C.2. CVS Server Software

- CVS RPM download (Red Hat Linux 7.1) ftp://speakeasy.rpmfind.net/linux/redhat/7.1/en/os/i386/RedHat/RPMS/cvs−1.11−3.i386.rpm
- Source tarball links can can be found at cvshome.org.

C.3. CVS Client Software

There is a variety of client access methods for using CVS on your development box.

- There are some great graphical clients for Linux, Windows, and Mac at http://www.cvsgui.org. These also give you the same access to all the command line cvs commands.
- jCVS is a great cross−platform graphical CVS client available at http://www.jcvs.org.
- jEdit is a great cross−platform text editor written in java, which not only has a CVS module that allows you to commit (upload) files directly from the editor, but also has a Interchange Tag Language (and Perl language) colorizer/parser. It is available from http://www.jedit.org.
