How To Use Mezzanine To Automate The Building Of RPM Packages For cAos Linux

Mezzanine is a set of software tools created to make building and maintaining software packages as simple and efficient as possible.

The Source Package Module Format

Mezzanine uses a simple storage format called SPM. SPM stands for Source Package Module, and the reason for this simple, yet useful, format is to bundle together (yet keep distinct) three separate items: the pristine source code, the patches applied to that source code, and instructions for building the package (the spec file for RPMs). The "source" is stored in a directory called "S", the "patches" are stored in a directory called "P", and the "spec file" is stored in a directory called "F" (for the "file" in "spec file", as "S" was already taken).

The format puts a directory with the same name as the package in the current directory. Inside the package named directory there are the three SPM format directories: "S", "P", and "F". If you check out the SPM from CVS, only directories that have files contents will be created (so the "S" source and "P" patch directories could be missing). After an "import" of an SRPM (source RPM) in Mezzanine's "local" mode, all three directories will be there. In the package named directory and it's subdirectories, Mezzanine's tools will allow fast and efficient patch and package creation, creating working directories there as needed.

Building A Package In Local Mode

If you do not have access to a CVS repository, or don't want one, you will have to use Mezzanine in local mode. This means that you will take a source RPM package and "import" it to the current working directory. This example will use the Mezzanine SRPM:

```
$ mzimport -L mezzanine-1.7-0.14.src.rpm
Importing /home/taj/RPM_BUILD/SRPMS/mezzanine-1.7-0.14.src.rpm
into mezzanine tree....
You requested local mode.  You will need to import the new tree
by hand (mzimport mezzanine).
$ ls
mezzanine
```

and now explore the contents of the newly created SPM directory structure:

```
$ cd mezzanine
$ find .
.
./S
./S/mezzanine.tar.gz
./P
./F
./F/mezzanine.spec
```

The source and the spec file are there, and if any patches were included in the SRPM, they would be in the "P" directory. There are none, so it is empty. I will fix that soon. :-)

Inside the package name directory ("mezzanine") execute the "mzprep" command:

```
$ mzprep
Creating working directory /home/taj/tmp/mz-local/mezzanine/work....
You may now chdir to work to make changes.
Use "mzpatch -n <patch_name>" to generate a patch when done.
$ ls
F   P   S   work
```

In this example I will make a small change:

```
$ cd work
$ ls
mezzanine
$ cd mezzanine
$ ls
bootable-cdtool  doc             mod      redhat-6.x-cdtool  specgen
buildtool        imgtool         perlpkg  redhat-7.x-cdtool  srctool
compstool        mediatool       pkgsort  revtool            templates
debian           mezzanine.spec  pkgtool  safetool           unofficial
uptool
$ echo "hi mom" > file.txt
```

but any number of modifications can be made in the working directory before creating a patch. Note that in this directory there is a spec file "mezzanine.spec". This file is included with the source for use by packagers who only download the gzipped tar archive, but it is not the spec file that will be used to build this package. That spec file sits alone in the "F" directory of this SPM.

Now that I have made the change, I will create the patch file using the "mzpatch" tool, but notice that I will have to include the "-L" argument because I am working in Mezzanine's "local" mode:

```
$ cd ../..
$ mzpatch -L -n mezzanine-1.7-hi_mom_file.patch
Creating working directory /home/taj/tmp/mz-local/mezzanine/work....
Created P/mezzanine-1.7-hi_mom_file.patch (5 lines).
You requested local mode.  You will need to add the new patch by
hand (mzadd P/mezzanine-1.7-hi_mom_file.patch).
```

The "mzadd" instruction would apply if we were going to include this patch in a CVS repository, but we will not. Fortunately, the patch will be included in the SRPM we will be building.

Now the spec file needs to be edited to include the patch:

```
$ vi F/mezzanine.spec
```

First, I need to find the "Source:" line and include a reference to my patch (mezzanine-1.7-hi_mom_file.patch) after it (and any other patches listed there):

```
    Source: %{name}.tar.gz
-> Patch1000: mezzanine-1.7-hi_mom_file.patch
```

```
   BuildRoot: /tmp/%{name}-%{version}-root
```

choosing a patch number that does not yet exist. The Mezzanine convention has been to start patch numbers at 1000 to separate patches created with Mezzanine from other patches previously include with a SRPM package.

Second, I need to include a "%patch" line for this patch in the spec file "%prep" stanza after the "%setup" line and any other "%patch" lines listed there:

```
   %prep
   %setup -n %{name} -T -c -a 0
-> %patch1000 -p1
```

making sure the patch number matches the one chosen in the "Patch#:" line ("Patch1000: mezzanine-1.7-hi_mom_file.patch" here), especially if the number is something other than "1000".

Third, add an entry under the "%changelog" line with the correct format for date, name, and email. Include only changes made to the spec file here, not changes made to the package as a whole:

```
   %changelog
-> * Tue Jun 08 2004 Troy Johnson <troj@caosity.org>
-> - added patch 1000
```

In addition to these three changes, you may want to update the "Release:" line, but do remember to put those changes into the "%changelog" entry under separate "-" bullets.

Save the changes to the spec file and exit the editor.

Now the package can be built using the "mzbuild" command:

```
$ mzbuild
...much package building output...
Package build succeeded.  Output files are:
mezzanine-1.7-0.14.src.rpm
mezzanine-1.7-0.14.noarch.rpm
```

The output files should be in the current directory (the package name directory, "mezzanine" in this example). Upload the SRPM to the cAos temple for submission to the autobuilder.

After the build there will be a "build.mezz" directory in the package name directory. Inside you will find the usual directories used to build RPM packages ("BUILD", "RPMS", "SOURCES", "SPECS", and "SRPMS").

If there are problems during the build process, try to divine whether the problems are related to Mezzanine, the build environment, or the particular package and spec file. Doing that will help troubleshoot the problem, and may guide you when you look for the best location to ask questions about it.

To get this example to build without trouble on my machine, I had to comment out a "BuildRequires:" line. I could have installed the required package, but chose not to do so. This problem could be addressed in other ways, and those ways will be explained here, but later on in the document.

Building A Package In CVS Mode

If you do have a CVS repository available for managing cAos packages, you will want to use this SCM (source code managment) to store and keep version information for your Mezzaninze SPMs.

Before you start, you will want to make sure you have your name in the GECOS field of "/etc/passwd" (on the machine you do CVS checkout on). This will make sure that your "Changelog" entries for CVS are identifiable by others.

If you don't have the CVS package installed, you can probably get it easily by executing this command as root:

```
# yum install cvs
```

After you have the CVS package installed, you can create a personal repository with these commands run as a normal user:

```
$ mkdir ~/cvs
$ export CVSROOT=~/cvs
$ cvs init
```

If you plan to use this repository all the time, put the "CVSROOT" environment variable export in your "~/.bashrc" file. The other commands need to be done only one time.

In case you do not have the package you wish to work with imported into your repository already, you will have to do that first:

```
$ mzimport mezzanine-1.7-0.14.src.rpm
Importing mezzanine-1.7-0.14.src.rpm into mezzanine tree....
cvs import: Importing /home/taj/taj/prog/cvs/cvsroot/mezzanine/S
N mezzanine/S/mezzanine.tar.gz
cvs import: Importing /home/taj/taj/prog/cvs/cvsroot/mezzanine/P
cvs import: Importing /home/taj/taj/prog/cvs/cvsroot/mezzanine/F
N mezzanine/F/mezzanine.spec
No conflicts created by this import
```

After you have done that, or if the package was in the repository already, you will have to "check out" the Mezzanine SPM package:

```
$ mzget mezzanine
cvs checkout: Updating mezzanine
cvs checkout: Updating mezzanine/F
U mezzanine/F/mezzanine.spec
cvs checkout: Updating mezzanine/P
cvs checkout: Updating mezzanine/S
U mezzanine/S/mezzanine.tar.gz
```

When you have checked the package out from CVS, you will notice the SPM directory structure is created with the addition of a "CVS" directory. The "CVS" directory is for cvs's use and so should be left alone until you are done interacting with CVS from this directory location.

Now you can modify the package, as was done in the Mezzanine "local" example:

```
$ cd mezzanine/
$ mzprep
```

```
Creating working directory /home/taj/tmp/mz-cvs/mezzanine/work....
You may now chdir to work to make changes.
Use "mzpatch -n <patch_name>" to generate a patch when done.
$ echo "hi mom" > work/mezzanine/file.txt
$ mzpatch -n mezzanine-1.7-hi_mom_file.patch
Creating working directory /home/taj/tmp/mz-cvs/mezzanine/work....
Created P/mezzanine-1.7-hi_mom_file.patch (5 lines).
cvs add: scheduling file `P/mezzanine-1.7-hi_mom_file.patch' for addition
cvs add: use 'cvs commit' to add this file permanently
Patch added and ready for commit.
$ vi F/mezzanine.spec
...add "Patch1000:", "%patch1000", and "%changelog" entries...
```

and then proceed to build the package:

```
$ mzbuild
...much package building output...
Package build succeeded.  Output files are:
mezzanine-1.7-0.14.src.rpm
mezzanine-1.7-0.14.noarch.rpm
```

After the package is built successfully, you can commit your changes to the CVS repository. Before you do, you should always clean up the the temp files and directoriesi. If you want to keep the RPM and/or the SRPM created with this last build, move it to another location now or it will be "cleaned" (removed). To clean up use this command:

```
$ mzclean
Cleaning and resyncing mezzanine...
Removing build.mezz...
Removing mezzanine-1.7-0.14.noarch.rpm...
Removing mezzanine-1.7-0.14.src.rpm...
Removing work...
Cleanup of mezzanine complete.
```

Now, with a clean directory tree and conscience, commit the modifications to the SPM to CVS with this command:

```
$ mzput
cvs update: Updating .
cvs update: Updating F
M F/mezzanine.spec
cvs update: Updating P
A P/mezzanine-1.7-hi_mom_file.patch
cvs update: Updating S
Please edit your commit message now...
cvs add: scheduling file `ChangeLog' for addition
cvs add: use 'cvs commit' to add this file permanently
cvs commit: Examining .
cvs commit: Examining F
cvs commit: Examining P
cvs commit: Examining S
RCS file: /home/taj/taj/prog/cvs/cvsroot/mezzanine/ChangeLog,v
done
Checking in ChangeLog;
/home/taj/taj/prog/cvs/cvsroot/mezzanine/ChangeLog,v  <--  ChangeLog
initial revision: 1.1
done
Checking in F/mezzanine.spec;
```

```
/home/taj/taj/prog/cvs/cvsroot/mezzanine/F/mezzanine.spec,v  <--
mezzanine.specnew revision: 1.2; previous revision: 1.1
done
RCS file:
/home/taj/taj/prog/cvs/cvsroot/mezzanine/P/mezzanine-1.7-hi_mom_file.patch,v
done
Checking in P/mezzanine-1.7-hi_mom_file.patch;
/home/taj/taj/prog/cvs/cvsroot/mezzanine/P/mezzanine-1.7-hi_mom_file.patch,v
<--  mezzanine-1.7-hi_mom_file.patch
initial revision: 1.1
done
```

The process spawns an editor to provide a CVS commmit message, and stores that in a file called "Changelog", and then commits that to CVS too. Make sure you keep your line lengths in the "Changelog" file to 72 characters or less (aligned with the last ")" character onthe first line) or it will look funny. The SPM directories and files are commited, and so now the changes are in CVS. Good job!

Since you have commited the changes, you can exit and delete this directory, or do some more changes and commit them with "mzput" too.

Building Packages In A Buildroot

First, a buildroot is a set of files made up from the smallest number of packages needed to build packages for a linux distribution (like cAos, for example). Build in a buildroot is preferred because it is a consistent, well known, minimal environment. Packages built there have a better chance of building correctly, with all "BuildRequires" correctly specified, because the minimal environment ensure that if they do not, they fail to build at all.

Because a buildroot can be rather large, and problems need to be fixed in it from time to time, it is usually available from somewhere via "rsync". The rsync source should be located and put in a script for easy execution, and perhaps put in a crontab so it will be brought up to date on a regular basis. Put the buildroot in a location with a lot of extra space:

```
$ rsync -arlHtSv temple.caosity.org::buildroot-2 /bigspace/caos2-br.dist/
```

the extra space is so that you can have a buildroot to do work in, and a pristine version to "rsync" from when you want to clean up the working buildroot after a package is built. You may also need a bunch of space to build and store the packages themselves. If you want a local working copy of the buildroot, execute this:

```
$ rsync -arlHtSv /bigspace/caos2-br.dist/ /bigspace/caos2-br.work/
```

and you will have a clean buildroot and and working buildroot. Now you can use the working buildroot to build packages.

Get everything done with the package (in Mezzanine CVS mode) up until the build part (command output is deleted here), keeping in mind that you will not have to do the "mzimport" step if you already have the "mezzanine" SPM in CVS:

```
$ mzimport mezzanine-1.7-0.14.src.rpm
$ mzget mezzanine
$ cd mezzanine
$ mzprep
```

```
$ echo "hi mom" > work/mezzanine/file.txt
$ mzpatch -n mezzanine-1.7-hi_mom_file.patch
$ vi F/mezzanine.spec
...add "Patch1000:", "%patch1000", and "%changelog" entries...
```

Now that everything is ready to build, "su" to root and use the working buildroot like this:

```
$ su
...type in root password at the prompt...
# mzbuild -r /bigspace/caos2-br.work
```

and Mezzanine will build the package in the buildroot environment instead of the current host's environment. Copy the RPM and SRPM products of the build out of the way and do the "mzclean" and "mzput" to finish up:

```
$ cp *.rpm ~/
$ mzclean
$ mzput
...enter Changelog information so commit will occur...
```

and you are done.

Buildroot Builds With Hints And BuildReqs

Mezzanine will allow you to install binary packages from a package repository that are required to build the current package, and do it automatically. The packages required to build a particular package are usually in the spec file on "BuildRequires:" and "BuildPreReq:" lines. To have Mezzanine install packages automatically, you must tell it how to do so:

```
# mzbuild -r /bigspace/caos2-br.work --hi 'yum -ty install'
```

and then afterward you will have to clean the buildroot using the pristine version you keep nearby:

```
# rsync -arlHtSv --delete /bigspace/caos2-br.dist/ /bigspace/caos2-br.work/
```

Mezzanine also allows for the use of "Hints". A "Hint" is similar to a "BuildReq" in format. It is not a requirement that a package mentioned in a "Hint" be installed for a package to build, but it should as a rule provide some advantage to the package being built.

An example of a "Hint" would be having the "MySQL-devel" package installed when building "postfix" allows "postfix" to access configuration data contained in a MySQL database. "postfix" will build fine without "MySQL-devel", so it isn't a "BuildReq", but it does gain an advantage when "MySQL-devel" is present.

"Hints" are currently implemented as a directory of files whose names correspond to packages. These files contain, one package name per line, "Hints" for the package that corresponds to the filename. So, the "postfix" file would contain a line with "MySQL-devel" on it, among others. To use the "Hints" you must get them with "rsync":

```
$ rsync -arlHtSv temple.caosity.org::hints-2 /bigspace/caos2-hints/
```

and keep them up to date (about the same frequency as the buildroot).

To use the "Hints", execute this command when building:

```
# mzbuild -r /bigspace/caos2-br.work --hi 'yum -ty install' -H /bigspace/caos2-hints
```

and Mezzanine will do it's best to add "BuildRequires", BuildPreReq", and "Hints" to the buildroot before building the current package.

It is also possible (and a good idea) to build as a user other than "root". This is accomplished using the "-u" command line switch as the "root" user:

```
# mzbuild -r /bigspace/caos2-br.work -u nobody
```

where "nobody" could be replaced with any user id. This can be combined with the installation and hint switches mentioned above:

```
# mzbuild -r /bigspace/caos2-br.work -u nobody \
      --hi 'yum -ty install' -H /bigspace/caos2-hints
```

to build the package as a non-root user in the buildroot, and install build requirement and hint packages as needed.

Parallel Builds

You can do parallel builds with Mezzanine by FIXME.

Remember, after doing a parallel build, execute this command:

```
# rm -rf /var/tmp/mezzanine*
```

to get rid of old temporary build directories FIXME.

Updating SPMs When New Sources Are Available

When a new source archive is available, check out the SPM module and copy the new source file into the correct directory:

```
$ cd ~/checkout
$ mzget mycoolpackage
$ cd mycoolpackage
$ rm S/mycoolpackage-1.1.tgz
$ cp ~/mycoolpackage-2.0.tgz S/
```

and update the spec file so it uses the new package source:

```
$ vi F/mycoolpackage.spec
```

and change the "Source" line to:

```
 -> Source: mycoolpackage-2.0.tgz
```

After this is complete, take a look inside the new source package and make sure any patches to the source still apply. If they do not, take the patches out of the SPM.

Before you commit anything back to the CVS repository, please try to build the package

and make sure everything is operating as it should:

```
$ mzbuild
```

You may want to install it somewhere and test the package a bit. If everything seems alright to you, check in the changes:

```
$ mzclean
$ mzput
```

and give yourself a pat on the back.

Creating SRPMs From Tar Archives

Mezzanine includes a utility called "specgen". "specgen" will do its best to create a working RPM spec file from a source archive. A directory will created in the current working directory, named for the package name. Inside that directory will be the spec file and the a copy of the source archive.

The spec file will contain a number of lines with the text "FIXME" on them, so the file will need to be edited before use. The "Summary:" and "%description" lines are impossible for "specgen" to guess, so read the documentation that accompanies the source and replace the "FIXME" information with the real stuff. "specgen" does not classify the RPM package group, so that will need to be replaced with good information also. Execute:

```
$ more /usr/share/doc/rpm-4*/GROUPS
```

to see a list of "Group:" possibilities a package can belong to.

The easiest way to turn the specfile and source into a SRPM or SPM is FIXME.

The PDR Format

Mezzanine can use other formats besides SPM, one of which is PDR. PDR means Package Development Repository and it is a storage format for the elements required to build RPM and SRPM packages, similar to default Mezzanine SPM format. The main difference between to the two formats is that while SPM separates the spec file, source, and patches into "F", "S", and "P" directories, PDR stores all three in the root of the packages named directory.

This may seem like a problem, but file naming conventions for spec files (typically "packagename.spec") and patches (should be "packagename-version#-what_patch_does.patch") serve to separate them from source files (everything else). So, choosing one format over another is more a matter of personal preference than anything else.

Problems And Troubleshooting

All Mezzanine perl programs with the "mz" prefix accept the "-d" command line argument. If you get some strange error message in response to a Mezzanine command, try the command again with the "-d" (for "debug") argument in addition to the ones already used. It should provide some useful information about what is going wrong.

Mezzanine Commands

Here is a command summary for Mezzanine utilities. It may contain errors and/or lies (no, really):

bootable-cdtool
creates bootable cdrom ISOs, needs an update

buildtool
builds RPM packages

compstool
verifies all needed packages are in comps file

imgtool
creates a root image directory, installing RPMs from a list

mediatool
creates an ISO file from a source directory

mzadd (revtool -a)
prepare a file for addition to CVS

mzann (revtool -q a)
make a note

mzannotate (revtool -q a)
make a note

mzblame (revtool -q a)
make a note

mzbuild (pkgtool)
build an RPM package from within a checked out SPM

mzci (revtool)
check in an SPM to CVS

mzclean (srctool)
cleanse an SPM of package files and temporary directories

mzco (revtool)
check out an SPM from CVS

mzdiff (revtool)
get a diff file smade form different versions of an SPM

mzget (revtool)
check out an SPM from CVS

mzimport (srctool)
create a new SPM module in CVS from a SRPM package

mzinfo (revtool)
get information about a version of an SPM

mzinst (pkgtool)
install a package

mzlog (revtool)
create a log entry for an SPM module in CVS

mzlogin (revtool)
login to a pserver CVS repository

mzmerge (srctool)
merge all changes made to an SPM into CVS

mzmv (srctool)
move a file within a SPM in CVS

mznew (revtool)
create a new file in a CVS SPM module

mzpatch (srctool)
create a patch from a working directory

mzpbuild (buildtool)
build a number of packages at the same time

mzpkg (pkgtool)
build a package from within an SPM

mzprep (srctool)
extract SPM sources and prepare to make a patch from a "work" directory

mzprod (buildtool)
set up multiple package build

mzprodbuild (buildtool)
start multiple package build

mzpurge (revtool)
dump all changes made to working directory

mzput (revtool)
check changes made to SPM module into CVS

mzreset (revtool)
dump all changes made to working directory

mzrm (revtool)
remove a file from a SPM module

mzrpm (pkgtool)
mzrtag (revtool)
mzstat (revtool)
mzstatus (revtool)
mzsync (srctool)
mztag (revtool)
perlpkg
pkgsort
pkgtool
redhat-6.x-cdtool
redhat-7.x-cdtool
revtool
safetool
specgen
srctool
uptool

Please contact troj@caosity.org or troj on #caos with errors, problems, and fixes for this document. Thanks for reading!