Dim User's Manual

Jean-Paul Chaput
Jean-Paul.Chaput@lip6.fr

June 2011

Contents

1 Introduction

The **Dim** package provides three tools: **Dim**, **yumport** and **yumsync**. **yumsync** and **Dim** are independant but **yumport** depends on **Dim**.

Thoses tools are mainly aimed at system administrators with a background in rpm/yum based distributions and rpm building.

Dim and **yumport** must be run on machine with the same architecture as the distribution. That is 32 bits for i386 and 64 bits for x86_64.

1.1 Getting Dim

Source and binary of **dim** rpm packages can be downloadeds from here:

source	dim-1.0.7-1.slsoc6.src.rpm
binary	dim-1.0.7-1.slsoc6.noarch.rpm

2 Dim

2.1 Goals of Dim

Starting from a medium size computer network, it is often useful to set up a local mirror of the distribution, so not to overload the bandwidth towards the outside of the

network and not saturate the public servers. Besides having a complete local copy can come in handy. When setting up the local mirror, the straighforward approach is to do a complete copy. Which means at least two repositories: the base distribution, then the security updates. As the repository holding the security updates contains all the previous outdated version of a package, it can grow big over time. Another aspect is that by keeping the repositories separated you have a "two step" install mode: first step install the base (outdated) distribution, then the updates are applieds. It is now less true because the installer (from SL6) automatically perform the update.

The second mirror approach is to merge the base distribution and it's updates and keep only the latest version of a package. The result is a distribution behaving like a *rolling* one, but only with the official updates.

This is mainly what **`Dim`** does: merge repositories and prune outdated packages. In addition, it embbeds **`createrepo`**, **`repoview`**, and can add GPG signature to packages that are lacking it.

2.2 Configuring `Dim`

Configuration is stored in `/etc/yum/dim.conf`. The file is in python format, allowing any legal python construct. The file consist in a series of call to the `addRepository()` method, which basically tells the repository into which we merge packages and from which repositories.

```
configuration.addRepository
   ( repoid     ="sl-local"
   , optionHelp ="Applies on SL 6 distribution repository."
   , comps      ="repodata/comps-sl6-%(basearch)s.xml"
   , rpmSubdir  ="Packages"
   , updateRepos=["sl-security"]
   )
configuration.addRepository
   ( repoid     ="sl-source-local"
   , optionHelp ="Applies on SL 6 source distribution repository."
   , updateRepos=["sl-source"]
   )
configuration.addRepository
   ( repoid     ="my-addons"
   , optionHelp ="Applies on <My Addons> repository."
   , updateRepos=[]
   )
configuration.addRepository
   ( repoid     ="my-addons-source"
   , optionHelp ="Applies on <My Addons> source repository."
   , updateRepos=[]
   )
```

addRepository() parameters	
repoid	The target repository id (into which we merge).
updateRepos	The list of repositories id from which to get the updates.
comps	A comps file for **createrepo** and **repoview** (optional). The %(basearch)s may be used to be architecture independant.
rpmSubDir	By default, **Dim** copies rpm in the same relative location as in the source repositories. But sometimes we need to insert an extra path, like in the base distribution.
optionHelp	The help message that will be displayed by **Dim** when help is requested

On the machine hosting the distribution and on which dim is run, there must be two sets of repositories.

1. The *normal* repositories, used for the host maintenance.
2. The repositories used by **Dim** to update the distribution.

Those two different sets must not be mixeds. To achieve the separation, **Dim** changes the root of the filesystem hierarchy from which yum operates. The **Dim** repositories are located in /home/dim/yum.root/etc/yum.repos.d/. For the same reason, to tune yum behavior when it's used by **Dim**, modify /home/dim/etc/yum.conf.

Using Dim

The **Dim** command line:

```
dim [--repoview] [--addsign] [--clean-cache] [--check-update] \\
    <repoid1> [repoid2 ...]
```

The default behavior of **Dim** is to merge the update and call **createrepo** on the selecteds repositories.

Dim parameters	
repoid	Specifies on which repository to run. This is the repoid given in the configuration file.
--repoview	Run **repoview** (not run by default).
--check-updates	Reports for avalaible updates. Do not merge them.
--addsign	Look for packages missing a signature and try to add one. This option disable the updating.
--clean-cache	Clear all cached yum data (equivalent to yum clean all).

Note 1: **`Dim`** will only look for update of package that are already present in the target repository. That is, if completly new packages appears in the source repository, they will not be added. You have to if manually the first time.

Note 2: Deprecated packages are not erased. They are moved into a backup directory, `/home/dim/backup/<repoid>/` so if something goes wrong they can be restored. This directory should be cleaned from time to time.

Other Uses

Another application is to create local repository that merge vendors `rpm`. You could for instance create a `vendor` repository that merges the contents of `Adobe` and `VirtualBox`.

3 `yumport`

`yumport` is an utility to ease the task of porting packages from sibling distributions like FEDORA or third party repositories like ATrpms. **`yumport`** is aimed at simplicity, it do not uses `chrooted` environment or sophisticated scheduled rebuilds and report systems.

`yumport` is able to recursively pull/install dependencies needed to build a given package. **`yumport`** is to perform the more tedious task of porting, but il will not correct the code for you or adjust dependency name changes. Nevertheless, many FEDORA packages rebuild without problems under SL 6 and it's for those that **`yumport`** has been created.

Using `yumport`

The **`yumport`** command line:

```
yumport --package=<package1> [--package=<package2> ...]
```

`yumport` parameters	
`--package`	The name of the package to be rebuild. Can be used multiple times.

If everything goes according to plan, the following cycle will be executed:

1. The source file is pulled from the source repository.
2. The sources are unpacked in the `rpmbuild` directory trees.
3. The dependency are looked upon. Either they are avalaible, and are installed, or they must be compiled from source. In the later case insert them in the head of the rebuild list and continue.
4. All dependencies are installed, the package is built.

5. Source (if needed) and binary packages are moved into the target repository.
6. **Dim** is called to update source & target repository.

Some important files and directories for **yumport**:

- `/etc/yum/yumport.conf` : the configuration file.
- `/home/dim/SRPMS.cache` : the directory holdind the original source files (downloaded from the source repositories).
- `/home/dim/rpmbuild` : the root directory of the `rpmbuild` process.
- `/home/dim/rpmmacros` : an on the fly generated file to configure `rpmbuild`.
- `/home/dim/yum.root/etc/yum.repos.d/` : the alternate `yum` directory where **yumport** get it's repositories (shared with **Dim**).

Configuring yumport

First part: Repository Configuration.

```
def guessFedoraDistTag ( release ):
    m = re.match(r".*\.fc(?P<fcversion>\d+)\..*",release)
    if m: return "fc%s" % (m.group("fcversion"))

    m = re.match(r".*\.(?P<disttag>[^.]+)$",release) # $
    if m: return m.group("disttag")

    return release

def guessPatchedDistTag ( release ):
    return "slsoc6"

configuration.addSourceRepo        ("my-addons-source",guessFedoraDistTag)
configuration.addSourceRepo        ("fedora-14-source",guessFedoraDistTag)
configuration.setTargetRpmsRepo    ("my-addons"       ,"RPMS")
configuration.setTargetSrpmsRepo   ("my-addons-source")
```

yumport configuration	
`addSourceRepo`	Add a source repository (by `id`). The second parameter must be a fonction indicating how to transform the `%dist` tag (see below).
`setTargetRpmsRepo`	The target binary repository, where the rebuild packages are to be put. The second parameter gives a subdirectory inside the repository.
`setTargetSrpmRepo`	The target source repository.

In order to keep track of the original source of a package, we customize the `%dist` tag thanks to the `guessFedoraDistTag()` function. The convention implemented here is to add up the target distribution tag (`slsoc6`) with the source one (say: `fc14`), the result being `.slsoc6.fc14`. You may implement any convention you like.

Second part: Dependency Resolution.

```
def binaryToSource ( binary ):
    def rePerlSepar ( mo ):
        if   mo.group(0) == '(': return "-"
        elif mo.group(0) == ')': return ""
        elif mo.group(0) == "::": return "-"
        return ""

    matched = True
    source  = binary
    if   binary.startswith("libXvMCW"):             source = binary
    elif binary.startswith("qt-webkit"):            source = "qtwebkit"
    elif binary == "timidity++-patches":            source = "fluid-soundfont"
    elif binary == "soundfont-utils":               source = "gt"
    elif binary == "gnu.regexp":                    source = "gnu-regexp"
    elif binary == "msv-msv":                       source = "msv"
    elif binary == "msv-xsdlib":                    source = "msv"
    elif binary == "jakarta-commons-configuration": source = "apache-commons-configuration"
    elif binary == "jaxp":                          source = "xml-commons-apis"
    elif binary == "xfce4-doc":                     source = "xfce-utils"
    elif binary.startswith("libgoom2"):             source = "goom2k4"
    elif binary.startswith("bitstream-vera"):       source = "bitstream-vera-fonts"
    elif binary.startswith("netbeans-platform"):    source = "netbeans-platform"
    elif binary.startswith("perl("):
        print source
        source = re.sub(r"\(|\)|::",rePerlSepar,source)
    else:
        matched = False
    return (source,matched)

# Put the lookup fonction in place.
configuration.setBinaryToSourceHook(binaryToSource)
```

While a package is not yet in the `yum` or `rpm` database, we cannot guess what it provides. The `binaryToSource` function is a work around for this problem. It behave like a lookup table, returning the right source package name for a give dependency. Most of the time, the name of the dependency match the name of the source package, but when it's not the case, we put an explicit translation. The function shown above has been successfuly tested to recompile `mplayer` and `vlc` from ATrpms.

Third part: tricky packages.

```
configuration.addExtraInfos("lash"   ,requires=["libuuid-devel"])
configuration.addExtraInfos("lirc"   ,builds  =[[] ,["--define", "kmdl_userland 1"]])
configuration.addExtraInfos("sdcc"   ,builds  =[["--define", "__strip /bin/true"]])
configuration.allowUpgrades(["libvpx", "libvpx-devel"])
```

addExtraInfos() parameters	
--requires	A supplemental dependency which is missing in the spec file.
--builds	A list of list. Allows to perform multiple builds with differents macro arguments. For instance lirc will be built twice: the first tile without any arguments (hence the []) and the second time with a -define "kmdl_userlan 1" arguments.

By default, **yumport** never override a package already present in the local repositories, but sometimes we have to allow an upgrade (this should be considered very carefully). allowUpgrages() takes a list of package that are alloweds to upgrade distributions ones.

4 tl2rpm

Automatic portage of CTAN packages. This script is mostly a PYTHON rewrite of tl2rpm.c from Jindrich NOVY. The main difference from the original program are:

- Package-oriented approach. Instead of generating all specs files from a complete TEX Live database (texlive.tlpdb), translate <package>.tlpobj.
- Fetch archive files from CTAN if needed.
- Rebuild the rpm package.

Configuration files of tl2rpm. They resides under /etc/yum/tl2rpm/.

File	Function
changelog	The changelog section to be inserted in every rpm (same for all).
cls.list	When looking inside .tex or .cls file for dependency, we cannot known the kind of required file. This list contains all the dependencies that are .cls, otherwise they are considered to be .sty.
pkg-CTAN.list	The almost whole list of components in the CTAN archive. You may feed it to tl2rpm with the --packages option to rebuild the whole archive (took almost two hours on my computer).
pkg-SOURCES.map	In some rare case, the name of the rpm package do not match the one of the CTAN component. The correspondances are given here (rpm name first, CTAN component second).
req-blacklist.map	Some dependency must not be took into account. They are listed in this file.
tl2rpm.conf	The main configuration file.

The `tl2rpm` command line:

```
tlrpm [--quiet] [--nocache] [--all-packages] \\
      [--max-fails=<number>][--report<report_file] \\
      [--packages=<package_list>] [package1 package2 ...]
```

yumsync parameters	
`--quiet`	Terse display. One line per package.
`--no-cache`	Ignore already downloaded archives in `rpmbuild/SOURCES`.
`--max-fails=<>`	The number of packages allowed to fail before we stop the run.
`--packages=<>`	A file containing a list of packages to rebuild. One package per line, comment start by "#".
`--all-packages`	Rebuild the whole CTAN archive.
`--keep-cache`	Keep previously cached datas. The default is to erase everything from a previous run. This is may not be convenient when updating big set of packages.
`--report=<>`	Write a report about what has been done in this file.

Note on package names: When giving a package name to `tl2rpm`, remove the `texlive-` prefix. Saves some typing...

5 yumsync

Yumsync allows you to keep a computer *striclty* synchronized on a predefined set of packages. It is different from the **distribution- synchronization** mode of `yum`. The `yum` mode ensure that versions of installed packages are keep in sync with the distribution repository, performing downgrade if necessary. It do not modify the set of installed packages (apart from the obsolete mechanism). The task of **yumsync** is to keep in sync the set of package itself.

Using yumsync

First, a schematic description as how **yumsync** work to help you interpret the displayed outpout. To avoid running on stalled cached `yum` datas, the cache is systematically cleared (equivalent to `yum clean all`, so every time you run **yumsync**, you will see the download of the repositories metadatas. Then, **yumsync** performs a first dependency solve run, using an empty `rpm` database to compute the exact set of packages to be installeds. And finally, it runs a second dependency solve on the system `rpm` database to align the installed package set on the requested one.

The **yumsync** command line:

```
yumsync [--conf=<configuration>] [--profile=<profileid>] \\
        [--nogpgcheck] [--sync]
```

yumsync parameters	
`--conf`	Uses an alternate configuration file
`--profile`	Force the profile on which to synchronize. If not specified try to match the host name against the host patterns of each profiles
`--sync`	Perform the actual synchronization. By default the script is in dry run mode
`--nogpgcheck`	Disable GPG checking
`--keep-cache`	Keep previously cached datas. The default is to erase everything from a previous run. This is may not be convenient when updating big set of packages.

Before performing the first synchronization you are strongly urged to run in dry mode and thorougly checks what packages are to be removed or installeds.

As the aim of **yumsync** is to synchronize each computer of a network, the configuration file has to be distributed over the network. To achieve this you may uses `--conf` to point on a configuration file located on a networked file system or the system configuration could be synchronized with `rdist`.

Defining Profiles — The Configuration File `yumsync.conf`

The configuration file is in python format, allowing for any python construct to be used. The configuration file define a list of profiles, each profile supplies a list of hostname patterns, a list of enabled repositories and a list of groups/packages to synchronize with.

```
configuration.addProfile ( profileid = "lepka"
                         , hosts     = [ "lepka"
                                       , ".*\.soc.lip6.fr" ]
                         , repoids   = [ "sl", "epel" ]
                         , packages  = [ "@base"
                                       , "@core"
                                       , "zsh"
                                       ]
                         )
```

Any number of profile can be defined. Host pattern may be overlaping, in this case, the first to match is used. A complete example of configuration file is given in subsection 5.

addProfile() parameters	
profileid	An unique identifier for the profile
hosts	A python list of regular expressions that will be matched against the host name
repoids	A python list of repository to be enableds while synchronising
packages	A python list of groups and/or single packages. Names should be kept from the comps file, like in kickstart.

A full dependency check is performed on the set of requested groups/packages using the yum depsolve mechanism. So you only have to put in the list the *top level* groups or individual package. A good starting point is the %package section of anaconda-ks.cfg file left after any installation.

Example of yumsync.conf

```
# -*- Python -*-
#
# The configuration file of yumsync.
# It's a Python file (*not* a module).
#
# Packages names/group names as in yum command line (pattern) format.

sl6Desktop = [ "@additional-devel", "@base", "@core", "@debugging", "@basic-desktop"
             , "@desktop-debugging", "@desktop-platform",  "@desktop-platform-devel"
             , "@development", "@directory-client", "@eclipse",  "@emacs",  "@fonts"
             , "@french-support",    "@general-desktop",    "@graphical-admin-tools"
             , "@graphics",        "@input-methods",        "@internet-applications"
             , "@internet-browser",   "@java-platform",   "@legacy-x",    "@misc-sl"
             , "@network-file-system-client",    "@office-suite",     "@performance"
             , "@perl-runtime",      "@print-client",      "@remote-desktop-clients"
             , "@scalable-file-systems", "@server-platform", "@server-platform-devel"
             , "@tex",            "@technical-writing",            "@virtualization"
             , "@virtualization-client",  "@virtualization-platform",  "@web-server"
             , "@console-internet", "@x11"
            # End of groups.
             , "elrepo-release", "epel-release", "adobe-release-i386", "atrpms-repo"
             , "sl-release-notes",   "zsh",   "xfig",   "graphviz",    "ImageMagick"
             , "inkscape",  "latex2html",  "thunderbird",  "createrepo",   "urlview"
             , "rsh",  "procmail",  "fetchmail",  "mutt",   "rdesktop",   "tigervnc"
             , "tigervnc-server", "gconf-editor", "vim-X11", "python-docs", "qt-doc"
             , "libXpm-devel",  "libXmu-devel",   "libXp-devel",   "openmotif-devel"
             , "rdist", "screen", "lm_sensors", "yum-plugin-versionlock"
            # End of individual packages.
             ]

configuration.addProfile ( profileid = "lepka"
                         , hosts     = [ "lepka", "shadock" ]
                         , repoids   = [ "sl", "elrepo", "epel" ]
                         , packages  = sl6Desktop + \
                                       [ "a2ps"               # From <r:epel>
                                       , "repoview"           # From <r:epel>
                                       , "fuse-sshfs"         # From <r:epel>
                                       , "mod_python"         # From <r:epel>
                                       , "trac"               # From <r:epel>
                                       ]
```

```
                        )

configuration.addProfile ( profileid = "sl6-64"
                        , hosts     = [ "sl6-64" ]
                        , repoids   = [ "sl", "elrepo", "epel" ]
                        , packages  = sl6Desktop + \
                                      [ "a2ps"               # From <r:epel>
                                      , "repoview"           # From <r:epel>
                                      , "fuse-sshfs"         # From <r:epel>
                                      , "mod_python"         # From <r:epel>
                                      ]
                        )
```