Package Management: A Hands-On Explanation

WORK IN PROGRESS

Anatomy of a Slackware package

A Slackware package is a simple TGZ or TXZ compressed archive containing:

- the tree structure of files and directories ;
- post-installation scripts ;
- the package description.

The name of every package provides a series of informations:

- the program name ;
- the program version ;
- the architecture of the package ;
- the build number.

Here's a few examples:

- `emacs-24.2-i486-1`
- `mozilla-firefox-15.0.1-i486-1`
- `vim-7.3.645-x86_64-1`

Managing Slackware packages using the traditional tools

Since its early releases, Slackware provides a collection of simple tools - the `pkgtools` - enabling the user to install, upgrade and remove software packages, as well as build them:

- `installpkg`
- `removepkg`
- `upgradepkg`
- `explodepkg`
- `makepkg`

Installing software packages

Install the Emacs editor from the Slackware DVD [1)]:

```
# mount /dev/cdrom /mnt/cdrom
# cd /mnt/cdrom/slackware/e
# installpkg emacs-24.2-i486-1.txz
```

```
Verifying package emacs-24.2-i486-1.txz.
Installing package emacs-24.2-i486-1.txz [ADD]:
PACKAGE DESCRIPTION:
# emacs (GNU Emacs)
#
# Emacs is the extensible, customizable, self-documenting real-time
# display editor. If this seems to be a bit of a mouthful, an
# easier explanation is that Emacs is a text editor and more. At
# its core is an interpreter for Emacs Lisp, a dialect of the Lisp
# programming language with extensions to support text editing.
# This version supports X.
#
# http://www.gnu.org/software/emacs/
#
Executing install script for emacs-24.2-i486-1.txz.
Package emacs-24.2-i486-1.txz installed.
```

If you're using the CD set, Emacs is on the first CD.

Checking if a package is installed

The package installation process has created a new entry in `/var/log/packages` :

```
# ls /var/log/packages/em*
/var/log/packages/emacs-24.2-i486-1
```

Knowing if a package is installed boils down to checking the existence of the corresponding entry in `/var/log/packages`. Example :

```
# ls /var/log/packages/*firefox*
/var/log/packages/mozilla-firefox-15.0.1-i486-1
```

Firefox is installed on the system, in version 15.0.1. Another example :

```
# ls /var/log/packages/kdebase*
ls: cannot access /var/log/packages/kdebase*: No such file or directory
```

There is no `kdebase-*` package installed on the system.

Removing a package

Use `removepkg` to remove an installed package. The command can take the simple basename of the package as an argument. Example:

```
# removepkg emacs
```

It's also possible to provide the complete name as an argument. In that case, it's better to call the command from within `/var/log/packages` and use tab completion:

```
# cd /var/log/packages
# removepkg emacs-24.2-i486-1
```

Upgrading a package

Slackware provides security updates for its latest releases. Visit the official site to know more about the latest updates:

```
# links http://www.slackware.com
```

1. Follow the `ChangeLogs` link.
2. Check out `Slackware-stable ChangeLog`.
3. Read the file `ChangeLog.txt` corresponding to the architecture of your system.

You can also use the Links browser to fetch updates manually. Before launching Links, create a `/root/updates` directory [2] to store your downloaded updates:

```
# cd
# mkdir updates
# cd updates/
# links mirrors.slackware.com
```

1. Follow the `Slackware File Tree` link.
2. Check out the directory corresponding to your release and architecture.
3. Change into the `patches/packages` directory.
4. Download any available updates.

Quit Links and install your updates like this :

```
# upgradepkg bind-9.9.1_P4-i486-1_slack14.0.txz

+==============================================================================
===
| Upgrading bind-9.9.1_P3-i486-1 package using ./bind-9.9.1_P4-
i486-1_slack14.0.txz
+==============================================================================
===
Pre-installing package bind-9.9.1_P4-i486-1_slack14.0...
Removing package /var/log/packages/bind-9.9.1_P3-i486-1-
upgraded-2012-11-21,12:14:32...
  --> Deleting /usr/doc/bind-9.9.1-P3/CHANGES
  --> Deleting /usr/doc/bind-9.9.1-P3/COPYRIGHT
  --> Deleting /usr/doc/bind-9.9.1-P3/FAQ
  ...
Verifying package bind-9.9.1_P4-i486-1_slack14.0.txz.
Installing package bind-9.9.1_P4-i486-1_slack14.0.txz:
PACKAGE DESCRIPTION:
```

```
bind (DNS server and utilities)
#
# The named daemon and support utilities such as dig, host, and
# nslookup.  Sample configuration files for running a simple caching
# nameserver are included.  Documentation for advanced name server
# setup can be found in /usr/doc/bind-9.x.x/.
#
Executing install script for bind-9.9.1_P4-i486-1_slack14.0.txz.
Package bind-9.9.1_P4-i486-1_slack14.0.txz installed.
Package bind-9.9.1_P3-i486-1 upgraded with new package
./bind-9.9.1_P4-i486-1_slack14.0.txz.
```

Another example :

```
# upgradepkg iptables-1.4.14-i486-2_slack14.0.txz
```

Know more about the contents of a package

Every package has a corresponding entry in `/var/log/packages`. These are all simple text files providing information about the contents of the respective packages. Example:

```
# less /var/log/packages/wget-1.14-i486-1
PACKAGE NAME:     wget-1.14-i486-1
COMPRESSED PACKAGE SIZE:     478.5K
UNCOMPRESSED PACKAGE SIZE:     2.0M
PACKAGE LOCATION: /var/log/mount/slackware/n/wget-1.14-i486-1.txz
PACKAGE DESCRIPTION:
wget: wget (a non-interactive network retriever)
wget:
wget: GNU Wget is a free network utility to retrieve files from the
wget: World Wide Web using HTTP and FTP, the two most widely used Internet
wget: protocols.  It works non-interactively, thus enabling work in the
wget: background after having logged off.
wget:
wget: The author of Wget is Hrvoje Niksic <hniksic@srce.hr>.
wget:
wget:
wget:
FILE LIST:
./
install/
install/slack-desc
install/doinst.sh
usr/
usr/bin/
usr/bin/wget
usr/man/
usr/man/man1/
usr/man/man1/wget.1.gz
```

```
usr/info/
usr/info/wget.info.gz
...
```

Managing Slackware packages with slackpkg

The `slackpkg` utility has been officially included in Slackware since the 13.0 release. It enables the user to manage Slackware packages much more comfortably.

A few remarks:

1. Only official Slackware packages are handled by `slackpkg`.
2. Third-party packages can be managed if you use Matteo Rossini's `slackpkg+` plugin.
3. Dependencies still have to be managed manually.

Initial configuration

Edit `/etc/slackpkg/mirrors` and comment out *one and only one* package source, for example:

```
# /etc/slackpkg/mirrors
...
# FRANCE (FR)
ftp://mirror.ovh.net/mirrors/ftp.slackware.com/slackware-14.0/
# http://mirror.ovh.net/mirrors/ftp.slackware.com/slackware-14.0/
```

If you are using a stable release of Slackware, don't get the section wrong and uncomment a mirror from `Slackware-current`. If you do that, you will upgrade to a development version of Slackware!

If you prefer managing packages locally without the benefit of updates, you can still use the Slackware installation DVD as a package source. In that case, you will have to configure the default mount point:

```
# /etc/slackpkg/mirrors
...
#------------------------------------------------------------------
# Local CD/DVD drive
#------------------------------------------------------------------
cdrom://mnt/cdrom/
...
```

Don't forget to mount the DVD before calling `slackpkg`:

```
# mount /dev/cdrom /mnt/cdrom
```

Update the information on available packages:

```
# slackpkg update
```

Note that the above command does not install any package updates. It only updates the internal list of packages you *can* install.

It's always a good idea to invoke `slackpkg update` before searching, installing or updating a package, so the system's informations about available packages are up to date.

Installing packages

Example with a single package:

```
# slackpkg install mplayerplug-in
```

Confirm the installation in the subsequent screen, and the package is automatically downloaded and installed.

You can also provide several packages as an argument:

```
# slackpkg install mplayerplug-in bittorrent
```

You can also manage whole package groups:

```
# slackpkg install kde
```

Another example for package groups:

```
# slackpkg install xfce
```

Remove packages

Example with a single package:

```
# slackpkg remove mplayerplug-in
```

As above, confirm the removal of the package in the subsequent screen.

Remove several packages at once:

```
# slackpkg remove mplayerplug-in bittorrent
```

Likewise, you can remove a whole package group:

```
# slackpkg remove kde
```

Or:

```
# slackpkg remove xfce
```

Upgrading packages

When a package update is available, you can install it using the following command:

```
# slackpkg upgrade iptables
```

Update several packages at once:

```
# slackpkg upgrade mozilla-firefox mozilla-thunderbird
```

It is common practice to keep your whole system up to date:

```
# slackpkg upgrade-all
```

Search for specific packages or files

Search for a specific package:

```
# slackpkg search k3b
Looking for k3b in package list. Please wait... DONE
The list below shows all packages with name matching "k3b".
[uninstalled] - k3b-2.0.2_20120226.git-i486-1
```

If the package is already installed, here's what you get:

```
# slackpkg search Terminal
Looking for Terminal in package list. Please wait... DONE
The list below shows all packages with name matching "Terminal".
[ installed ] - Terminal-0.4.8-i486-1
```

You can also search for individual files. The search will eventually display on or several packages containing the file in question:

```
# slackpkg file-search libncurses.so
Looking for libncurses.so in package list. Please wait... DONE
The list below shows the packages that contains "libncurses\.so" file.
[ installed ] - aaa_elflibs-14.0-i486-4
[ installed ] - ncurses-5.9-i486-1
```

If you want to know more about the content of a package:

```
# slackpkg info mesa

PACKAGE NAME:  mesa-8.0.4-i486-1.txz
PACKAGE LOCATION:  ./slackware/x
PACKAGE SIZE (compressed):  19208 K
PACKAGE SIZE (uncompressed):  83930 K
PACKAGE DESCRIPTION:
mesa: mesa (a 3-D graphics library)
mesa:
mesa: Mesa is a 3-D graphics library with an API very similar to that of
mesa: another well-known 3-D graphics library.  :-)  The Mesa libraries are
mesa: used by X to provide both software and hardware accelerated graphics.
mesa:
mesa: Mesa was written by Brian Paul.
mesa:
```

Cleaning the system

Remove all third-party packages:

```
# slackpkg clean-system
```

If you decide to keep some of the packages, simply unselect them in the subsequent screen.

You can also use `slackpkg` to repair a damaged package. Let's say I accidentally deleted the file `/usr/bin/glxgears`. First, I have to search for the package providing that file:

```
# slackpkg file-search glxgears
Looking for glxgears in package list. Please wait... DONE
The list below shows the packages that contains "glxgears" file.
[ installed ] - mesa-8.0.4-i486-1
```

With this information, I can simply reinstall the package:

```
# slackpkg reinstall mesa
```

Rebuild official packages

Slackware provides the entire system's source code in the `source` directory. Every binary system package will have his corresponding source directory. These source directories usually contain:

- the source code for the application or the library;
- its fabrication recipe in the shape of a `*.SlackBuild` file;
- the package description in a `slack-desc` file;
- eventually, a post-installation script named `doinst.sh`;
- various other files like patches, custom menu entries, etc.

Build a package from source

In the example below, we will build the `Terminal` application from the source code provided by Slackware. You might want to remove the corresponding package if it is installed.

The `Terminal` package is Xfce's terminal. In Slackware 14.1, the package has been renamed to `xfce4-terminal`.

```
# removepkg Terminal
```

Choose an appropriate place on your system to store the source code and the scripts, for example:

```
# cd
# mkdir -pv source/Terminal
mkdir: created directory 'source'
mkdir: created directory 'source/Terminal'
# cd source/Terminal/
# links mirrors.slackware.com
```

Fetch the content from the `source/xfce/Terminal` directory on a Slackware mirror. Here's what we get:

```
# ls -lh
total 1,4M
-rw-r--r-- 1 root root  821 nov.  24 15:09 slack-desc
-rw-r--r-- 1 root root 1,4M nov.  24 15:11 Terminal-0.4.8.tar.xz
-rw-r--r-- 1 root root 3,6K nov.  24 15:10 Terminal.SlackBuild
```

Make the `Terminal.SlackBuild` file executable and start the building process:

```
# chmod +x Terminal.SlackBuild
# ./Terminal.SlackBuild
```

The script initiates the package compilation. If everything goes as expected, the operation exits with the following message:

```
Slackware package /tmp/Terminal-0.4.8-i486-1.txz created.
```

Now we can install the resulting package:

```
# installpkg /tmp/Terminal-0.4.8-i486-1.txz
```

Modify an official Slackware package

The main reason for rebuilding an official package is to modify it, for example to add or strip certain functionalities. In the following example, we will rebuild the `audacious-plugins` package in order

to modify the Audacious audio player. The vanilla application sports two different graphical interfaces, and we will disable one of them.

Let's begin with removing the package if it is installed:

```
# removepkg audacious-plugins
```

Now create a suitable directory to store the source code:

```
# cd /root/source
# mkdir audacious-plugins
# cd audacious-plugins
# links mirrors.slackware.com
```

Fetch the contents of the `/source/xap/audacious-plugins` directory and make the `audacious-plugins.SlackBuild` script executable:

```
# chmod +x audacious-plugins.SlackBuild
# ls -lh
total 1,4M
-rw-r--r-- 1 root root 1,4M nov.  24 15:28 audacious-plugins-3.3.1.tar.xz
-rwxr-xr-x 1 root root 4,0K nov.  24 15:28 audacious-plugins.SlackBuild*
-rw-r--r-- 1 root root  892 nov.  24 15:28 slack-desc
```

Now edit `audacious-plugins.SlackBuild` and add one option:

```
...
# Configure:
CFLAGS="$SLKCFLAGS" \
CXXFLAGS="$SLKCFLAGS" \
./configure \
  --prefix=/usr \
  --libdir=/usr/lib${LIBDIRSUFFIX} \
  --sysconfdir=/etc \
  --mandir=/usr/man \
  --enable-amidiplug \
  --disable-gtkui \          -> add this option
  --program-prefix= \
  --program-suffix= \
  ${ARCHOPTS} \
  --build=$ARCH-slackware-linux
...
```

Build and install the package:

```
# ./audacious-plugins.SlackBuild
...
Slackware package /tmp/audacious-plugins-3.3.1-i486-1.txz created.
# installpkg /tmp/audacious-plugins-3.3.1-i486-1.txz
```

Choosing your configuration options for compiling

The source configuration script (or more exactly the sometimes very long line in the SlackBuild beginning with `./configure`) often displays an overview of activated and/or deactivated options. To interrupt the package construction process and display this overview, you can temporarily edit the SlackBuild like this:

```
...
# Configure:
CFLAGS="$SLKCFLAGS" \
CXXFLAGS="$SLKCFLAGS" \
./configure \
  --prefix=/usr \
  --libdir=/usr/lib${LIBDIRSUFFIX} \
  --sysconfdir=/etc \
  --mandir=/usr/man \
  --enable-amidiplug \
  --program-prefix= \
  --program-suffix= \
  ${ARCHOPTS} \
  --build=$ARCH-slackware-linux

exit 1          -> add this option to interrupt the script

# Build and install:
make $NUMJOBS || make || exit 1
make install DESTDIR=$PKG || exit 1
...
```

Now run the script and wait a few seconds for the configuration overview:

```
# ./audacious-plugins.SlackBuild
...
Configuration:
...

Interfaces
----------
GTK (gtkui):                            yes
Winamp Classic (skins):                 yes
```

Use the `./configure –help` option to display a list of all the possible options:

```
# tar xvf audacious-plugins-3.3.1.tar.xz
# cd audacious-plugins-3.3.1
# ./configure --help | less
...
--disable-speedpitch    disable Speed and Pitch effect plugin
--disable-gtkui         disable GTK interface (gtkui)
--disable-skins         disable Winamp Classic interface (skins)
```

```
--disable-lyricwiki     disable LyricWiki plugin (default=enabled)
...
```

The SlackBuild script already takes care of automatically uncompressing the source tarball to the `/tmp` directory. So you can simply run `./configure –help | less` from this directory, without manually uncompressing the source tarball to the current directory.

In the present case, activating certain functionalities like for example managing proprietary audio formats will depend on the presence of the corresponding libraries on your system.

Once you've chosen all your configuration options, get rid of the temporary `exit 1` command in your script and launch the build and installation process:

```
# ./audacious-plugins.SlackBuild
...
Slackware package /tmp/audacious-plugins-3.3.1-i486-1.txz created.
# installpkg /tmp/audacious-plugins-3.3.1-i486-1.txz
```

Building third-party packages

Slackware offers only a limited choice of packages compared to behemoth distributions like Ubuntu or Debian. More often than not, you'll want to install a package that's not provided by the distribution. In that case, what can a poor boy do?

The SlackBuilds.org website is probably the best address to find third-party software. You won't find any packages there, because SlackBuilds.org is *not* a binary package repository nor will it ever be. It's an extremely clean and well organized collection of build scripts, each one reviewed and tested. Using these scripts will enable you to build about every piece of third party software under the sun.

Building packages using the SlackBuilds.org scripts

In the following example, we will build and install the `cowsay` package using the build script provided by SlackBuilds.org.

For a start, `cd` into the build directory we've defined earlier:

```
# cd /root/source
```

Download the following components into this directory :

1. the compressed tarball containing the scripts to build the package;
2. the compressed source code tarball.

In our case:

```
# links http://slackbuilds.org
```

1. In the Search field in the upper left corner of the screen, type `cowsay`, move the cursor to `Search` (CursorDown key) and confirm by hitting Enter.
2. Follow the `cowsay` link on the search results page.
3. Once you're on the `cowsay` page, download the SlackBuild (`cowsay.tar.gz`) and the source code (`cowsay-3.03.tar.gz`) and quit Links.

Alternatively, use `lynx` instead of `links`.

Here's our two downloaded tarballs:

```
# ls -l cowsay*
-rw-r--r-- 1 root root 15136 nov.  25 08:14 cowsay-3.03.tar.gz
-rw-r--r-- 1 root root  2855 nov.  25 08:14 cowsay.tar.gz
```

Uncompress the tarball containing the scripts:

```
# tar xvzf cowsay.tar.gz
cowsay/
cowsay/cowsay.SlackBuild.patch
cowsay/README
cowsay/slack-desc
cowsay/cowsay.SlackBuild
cowsay/cowsay.info
```

Eventually, you can do a little cleanup and delete the tarball:

```
# rm -f cowsay.tar.gz
```

Now move the source tarball to the newly created `cowsay/` directory:

```
# mv -v cowsay-3.03.tar.gz cowsay/
« cowsay-3.03.tar.gz » -> « cowsay/cowsay-3.03.tar.gz »
```

Here's what we have:

```
# tree cowsay
cowsay
|-- cowsay-3.03.tar.gz
|-- cowsay.info
|-- cowsay.SlackBuild
|-- cowsay.SlackBuild.patch
|-- README
`-- slack-desc
```

Now `cd` into that directory. Check if the `cowsay` SlackBuild is executable, and then launch it to start

the package construction:

```
# cd cowsay/
# ls -l cowsay.SlackBuild
-rwxr-xr-x 1 kikinovak users 1475 mai   27  2010 cowsay.SlackBuild*
# ./cowsay.SlackBuild
...
```

If everything goes well, the process spews out a package in `/tmp`, or more exactly in the `$OUTPUT` directory defined by the script:

```
...
Slackware package /tmp/cowsay-3.03-noarch-1_SBo.tgz created.
```

All that's left to do is install the package using `installpkg`:

```
# installpkg /tmp/cowsay-3.03-noarch-1_SBo.tgz
# cowsay Hi there !
 -------------
< Hi there ! >
 -------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Managing package dependencies

Some packages require the presence of other packages, either to build (*build dependencies*) and/or to run (*runtime dependencies*) correctly. In some cases, a required package can depend itself on one or more other packages, and so on.

To take an example, let's have a look at the `libgnomeprint` page on SlackBuilds.org. The package description is followed by the following caveat:

```
This requires: libgnomecups.
```

Moreover, every script tarball contains an `*.info` file which states explicitly all the required package dependencies. If we look at the `libgnomeprint.info` file, we'll find a REQUIRES field:

```
PRGNAM="libgnomeprint"
VERSION="2.18.8"
HOMEPAGE="http://www.gnome.org"
...
REQUIRES="libgnomecups" ----> package dependency
...
```

The REQUIRES field has been introduced with Slackware 14.0.

This simply means that before we build the `libgnomeprint` package, we have to build and install the `libgnomecups` package.

Besides strictly required dependencies, a package can also have some optional dependencies to offer some extra functionality. As an example, the Leafpad text editor can be built against the optional `libgnomeprint` and `libgnomeprintui` dependencies.

WORK IN PROGRESS

Sources

- Originally written by Niki Kovacs

slackware, package, management, author kikinovak

1)
if it's not already installed

2)
This is just an example, of course. Feel free to use any convenient place on your system.