Slackware Package naming convention

- Author: Stuart Winter <mozes at slackware.com>
- Date….: 18-Mar-2020
- Version : 1.00
- Purpose : Provide a best practice for naming packages for Slackware ARM.

ARM package naming convention

Exposition of the package name format:

```
  foo-1.0-arm-1.txz
  foo=the package name
  1.0=the version of the package (as set by its authors)
  arm=the target architecture for which this package is built
  1=the build number
  txz=file extension of a compressed tar archive, using the 'xz' compression
utility.
```

In Slackware ARM, the 'arm' label is **relative** to the particular release of Slackware ARM; it simply means that the software is compiled to target the baseline architecture, as configured in the gcc package:-

On a Slackware ARM 15.0 release:

```
root@zippy:~# gcc -Q --help=target | egrep '^. -(march|mfloat-abi)='
  -march=                               armv7-a+fp
  -mfloat-abi=                          hard
```

```
On a Slackware ARM 14.2 release:
root@zaden-142:~# gcc -Q --help=target | egrep '^. -(march|mfloat-abi)='
  -march=                               armv2
  -mfloat-abi=                          soft
```

Note: this is **supposed** to be 'armv5te' - something must have gone wrong in the last gcc package! However, since the build scripts set -march=armv5, it's not an issue.

This is no different from having the following package for Slackware on 32bit x86 architecture:

```
  foo-1.0-i586-1.txz
```

You could not determine - from the package name alone - whether this was from Slackware 14.2 or Slackware 15.0!

Exceptions

In Slackware ARM, the only exception to this rule is when a package could build, but only for a target architecture that was greater than the baseline.

For example, in Slackware ARM 14.2, we find the following package available within the 'unsupported' directory on the FTP site.

```
 'mozilla-firefox-34.0.5-armv6j-1_slack14.1.txz'
```

This was made available **outside** of the main tree for two reasons:

1. It had to be built on a machine that wasn't armv5. All build machines must be the baseline target - otherwise it can cause problems where packages are built that *don't* run on the baseline (due to auto-detection of platform features at build time).
2. Ordered List ItemThe package cannot run on a machine whose architecture version is ⇐armv6j

and as such, shouldn't be within the main Slackware ARM distribution tree.

In this instance, armv6j was the lowest architecture that the package would build for: the Firefox developers had already abandoned support for armv5 hardware because (IIRC) they wanted the hardware support within the newer architecture, to render web pages/media, etc.

With these exceptions, the architecture element of the package reflects the target architecture to make it clear that this package requires an ARM-based computer where the architecture is a minimum of armv6j.

But what about soft/hard float!?

Essentially the ABI defines the structures and conventions about how the components of the OS (binaries/libraries, etc.) interact.

Slackware ARM has been through three ABI (Application Binary Interface) changes:

Releases older than 12.2 used (what Debian call) the 'legacy' or 'old' ABI. This was the first ABI for ARM.

- Slackware ARM 13.37 up to and including 14.2 were built against the 'software floating point' ABI.
- Slackware ARM 15.0 and greater are built against the 'hardware floating point' ABI.

It doesn't matter what the ABI is called: it just matters that users know which version of Slackware ARM your package is targeting, because each of the ABIs is incompatible with one another.

In the Slackware x86 source tree, some of the SlackBuild scripts - when built on armv7 hardware - may cause the architecture element of the package name to be 'armv7'. However, this is legacy code for alienBOB's work on an earlier ARM hardware floating point port.

The reason it isn't helpful to name the architecture element of the package as anything other than 'arm' (apart from the exception noted above), is due to the ABI. You could build the same package on three releases of Slackware ARM (one with legacy ABI, one with soft ABI, one with hard), set the

CFLAGS to be -march=armv3 (so that the compiler emitted code that could run on all ARM architectures that were ever supported by Slackware ARM), and name it thusly:

```
  foo-1.0-armv3-1.txz
```

And despite every ARM computer currently running in the world (presumably) being able to execute ARMv3 instructions, it'd fail because the ABI is different.

Apart from in relation to the exceptions (as apreviously noted), it's simply an unhelpful and useless distinction.

Recommendations for naming packages that you intend on distributing

When distributing packages you cannot assume that even if the ABI remains constant across Slackware releases, that your package would work on more than one Slackware release.

This is because the environment around it changes! If your package is dynamically linked, it may be linked to multiple libraries, which are either not present within that Slackware release, or are at different major versions and as such may have a different API (Application Programming Interface).

Therefore, the most simple naming scheme recommendation is to adopt how the Slackware patches are named:

```
 foo-1.0-arm-1_slack14.2.txz
```

This way it's clear that:

- The architecture is 'arm', not i586, x86_64, aarch64
- Is targeting Slackware 14.2.

But I'm distributing packages for -current!

In which case you'd be better with an additional tag, such as

```
 foo-1.0-arm-1_slack14.2+.tgz
```

Appending the '+' to the version of Slackware is the convention present within /etc/slackware-release, whenever a release of Slackware is made and the new development branch ('-current') begins.

This is on a machine running the Slackware 14.2 release:

```
root@zaden-142:~# cat /etc/slackware-version
Slackware 14.2
```

This is on a machine, running Slackware-current, at a point in time where the previous release of Slackware was 14.2. Note the '+' suffix:

```
root@zippy:~# cat /etc/slackware-version
```

```
Slackware 14.2+
```

This is better than trying to guess what the next version of Slackware might be and setting your build number to, say, '-1_slackpre15.0'.

Also, it might not be wise to distribute binary packages for -current because you will continuously need to check whether they function, as the development of Slackware-current rolls on (refer to the discussion about libraries above).

I hope that helps!

Sources

The original document is stored here.

howtos, arm, author , mozes