Ruarí's thoughts

Ruarí Ødegaard works for Opera and has been making Opera packages available for Slackware for a long time. The article below was posted on his blog in 2011 and has attracted lots of attention since. Now that his employer has ended the “My Opera” pages this article has been conserved on archive.org but it deserves a place on the Slackware Documentation Wiki as well.

Package and dependency management shouldn't put you off Slackware

Since I have been running Slackware as my primary distro I've heard the occasional comment along the lines of, “Manual dependency management must be a real hassle” or worse, “How do you work without a package manager?”. Rather than repeat my response ad nauseam individually to those asking such questions, I thought I would save myself the trouble and type up my thoughts once publicly, so that I could simply point them to this post. 

Package management

Let's start with the second suggestion first, because it is totally false. Slackware does have a full set of tools for managing packages. The only obvious 'missing feature' is dependency management, which some people incorrectly equate with package management. I'll explain why this one seemingly important missing feature isn't as big a problem as you might think a little later. First, lets remember that package management is actually a much broader subject and also encompasses many other important tasks such as download, installation, logging/tracking, upgrading, removing and package creation. The package management tools provided by Slackware perform all of these other tasks with aplomb and even provide several advantages over some of the more popular Linux package managers.

What are these advantages? Well firstly, all logs are written in plain text, in a way that is easily read by humans but still machine-parseable. This means I am not just limited to the Slackware provided tools. I can run queries against them with a wide variety of standard UNIX utilities directly. I can read logs with less, cat or any text editor, and I can process, manipulate and collate packaging data with grep, awk, python, perl, etc. Or to put it another way, I am not forced to use any distro specific tool to gather (or even change), the information stored about my package setup.

Similarly the packages themselves are equally open, since they are simply compressed tarballs containing the filesystem contents and some optional text files that describe the package and perform any needed post-installation steps. You can view, open and even edit packages with a wide selection of tools (GNU tar, BSD tar, perl, etc.). Indeed if you really wanted to you could even create them without using Slackware's provided makepkg tool [1)]. Though I must strongly stress that last point really isn't recommended, as admittedly there are a couple of potential 'gotchas' related the exact listing of the internal file path and dealing with symlinks. Still, it is possible! For comparison consider the RPM format, which effectively requires installing rpm and associated tools for any reasonable level of query or manipulation. Installing rpm on non-rpm based systems can range from

being a minor inconvenience to being an almighty hassle, or even down right impossible, the further you move beyond the world of Linux-based OSes. Even deb, which is vastly better in this regard, still uses the relatively obscure (outside of Unix-like systems) ar format as its outer container.

Finally, perhaps the most interesting difference is that all of the packaging tools are implemented as simple shell scripts. This greatly lowers the barrier of entry to tweaking them, bringing it within the bounds of a user with intermediate to advanced Linux knowledge, rather than a Linux 'Guru'. For example, prior to the change of default Slackware package compression format (from gzip to xz) it was not uncommon for some Slackers to hack in support for alternative compression formats themselves. How many deb or rpm users (not developers) do you know who have hacked and adjusted their package managers to better suit their own personal needs or tastes?

I think the point to take away from all this is that one of the upsides of using a system like Linux, is that it is supposed to empower users and give them greater control of their system. In this regard Slackware's package management tools excel.

Why dependency management doesn't have to be difficult

So … dependencies!  This is probably the best known difference between Slackware and other popular distributions. Slackware doesn't even attempt to automatically handle dependencies during package install, leaving it up to the user to figure out what combination of other packages must be installed. From the outside and recalling some of my own preconceptions prior to moving to Slackware, I can see how this would appear to be a major omission.

One of the main reasons that Slackware doesn't bother with automated dependency management is that it doesn't need to. The person installing can do that and it doesn't even have to be that hard. Whilst it is possible to completely customise the selection of installed packages, users who are new to Slackware are recommended (by the accompanying documentation) to start with a full install (i.e. every package). And in fact, there are many seasoned Slackware users who also do this, even if they are fully capable of pairing down the package selection to something more minimal. Obviously a full install means that dependencies are completely satisfied. See, I told you Slackware was simple! 

I realise a few people might now be thinking, “Well isn't that just too simple, treating the distro like a monolithic system, rather than the modular entity it is?”. I agree, indeed it makes you wonder why every other distro had to make things so hard. 

Now of course some people reading this are probably already thinking up a few potential problems with this approach. I can make a fair guess at what these potential problems are because I have heard them before. There two most common arguments I have seen for not wanting to do a full install are, “it wastes space” and “it bloats the system, slowing it down”. The reason for assuming these will be issues, is often based on experiences with other distros. Once you understand a little bit more about Slackware's way of doing things it soon becomes obvious that neither are likely to have much impact on you if you choose to use Slackware.

Lets start with wasting space. According to the back cover of the official install DVD of version 13.37

(the current stable release as I write this), a full install of Slackware occupies 6.8Gb. Whilst this might seem like a lot to those of us who have been using computers for a number of years, in modern terms it is almost nothing. Consider the fact that even amazon.com (not the cheapest place for electronics) sells decent 500Gb disk drives for less than $50 (USD), meaning that 6.8Gb of space costs less than $0.70! Now I will admit that a hard disk is one of the cheapest ways of adding storage and isn't always feasible on smaller laptops or netbooks. However on the other hand, the cost is still very low even when choosing more expensive ways of adding storage to your system. For example, Amazon also offers new 8Gb microSD cards for less than $5 (USD). So you have to consider is it really worth your time to even consider cutting down the install selection, at least from a spacing saving perspective?

This obviously begs the question, “so wouldn't you always do a full install for all distros?”. There are two parts to this. Firstly some distros (such as Debian or openSUSE) have very large repositories, packing just about every open source application available (including a lot of stuff that only a handful of people will never even consider using). So if you did this you really would start to waste a lot of disk space. Perhaps more importantly the system really would start to slow down due to all the bloat.

On Slackware a full install running a lightweight window manager like Fluxbox will run every bit as fast a slimmed down install running Fluxbox. Why? Because of its simple design philosophy. Slackware does not make any assumptions about what the user intends to do with his or her software. This is unlike the majority of other distros, which do make assumptions and try to automate as much as possible in line with those assumptions.

This means that on most distros if you install some application that runs as a daemon/service, the install scripts in the post install of that package will typically start it immediately after installation finishes. This would appear to make good sense as why would you be installing it otherwise? Whilst it is indeed most likely that you would want it started, there are other possibilities and for this reason Slackware doesn't attempt to second guess the answers to questions like these, leaving it up to you. For example, you might not want it running all the time or you might have installed the package only to access one of the other binaries or libraries installed alongside the main program. Who knows, you might even have installed it because you couldn't find documentation on the author's website and wanted to read the included files or man page to find out more. The result of all this is that unless you enabled a service during the Slackware install, or later manually, most of the software installed just sits on your disk doing nothing other than taking up space and hence has no affect on the speed of your running system at all.

So yes, a large amount of software may be left just sitting around doing nothing, but you know what? It has little (if any) real negative affect on me, so why would I care? Indeed you never know when one of these applications might turn out to be useful. I don't typically want to run a full featured web-server on my desktop but should I ever need to test something that might require it, I have Apache installed and ready to fire up! Indeed this simple way of not having to think about dependencies can sometimes be a real advantage. If I want to try some program or utility at some point in the future I quite often find it is already there, negating the need to search via the package manager and then install it. This is particularly nice when installing an application would have pulled down hundreds of Mbs of extra dependencies. I might not care about some used disk space but I do care about wasting my networking connection if I am in the middle of using it or I am on a slow network.

And what about dependencies for non-official packages? Well most of the popular third party/non-official Slackware repositories provide some means to simplify dependency resolution, sometimes this is done by providing a tool to deal with the inter-dependencies found within that repository, or other times by simply listing any non-official (by that I mean not included in the standard repository) dependencies in README files provided with each package. This is the approach taken by what is

probably the most popular third party repository these days, SlackBuilds.org. It works because Slackware installs such a good selection of common libraries, which means that many of the applications already have the majority (if not all) of their dependencies fulfilled by a complete Slackware install. Since very few extra packages are needed it doesn't tend to be the hassle many might expect. And for the really lazy, using helper tools such as sbopkg (and its queuefiles system) you can even fully automate the process.

So there you go, not only does Slackware have a decent set of package management tools, the one area that tends to scare people off (dependency management) doesn't have to be daunting at all!


Sources

- Original source: Ruario's archived blog post of Monday, September 26, 2011
- Originally written by Ruarí Ødegaard

howtos, sbopkg, dependencies, slackbuilds, m2nb, author ruario

1)

If you really did want to create your own Slackware packages without makepkg (for whatever reason), here is a short summary of how it could be done. Switch to the root of your unpacked package contents and check for symlinks: “find . -type l”. If they exist convert them into shell script code appended to doinst.sh: “find * -type l -printf '(cd %h ; rm -rf %f)\n(cd %h ; ln -sf %l %f)\n' -delete » install/doinst.sh”. Finally, create a compressed tar archive with version 1.13 style directory formatting, either by using GNU tar-1.13 directly (it is installed on Slackware) or by forcing modern GNU tar to emulate this style of formatting by using a transform: “tar caf /tmp/some-application-1.0-x86_64-1.tgz . –xform='s,^\./\(.\),\1,' ”