JOIN THE SLACK SIDE

Forget your Ubuntus and Fedoras – true Linux Nirvana comes from following the way of Slackware, according to **Andrew Conway**.

Back in the day, before there were friendly GUI installers; before pretty became a feature, and before a Linux-based OS conquered the world through the magic of smartphones, there were only a handful of Linux distros. Most of them have fallen by the wayside over the years – who remembers Yggdrasil, Caldera or Linspire?

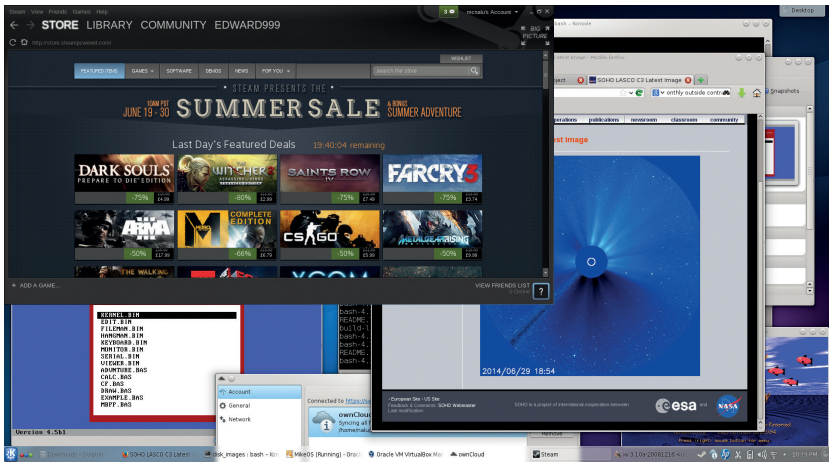
There is one survivor from those days though, and you can try it today: Slackware. Slackware is simple. By that, do we mean that it's is easy to install, set up and use? If you charge in armed only with enthusiasm, then the answer is probably no. But if you have some experience with Linux, are prepared to read the documentation and invest enough time in understanding it, then Slackware can be easy in all of these ways and give you a rock-solid, modern operating system that's under your full control.

The simplicity of Slackware is in how it operates, the way packages are handled and how key parts of it are written and configured using text files. When it is set up, it can be as easy to use as any other distro with a full-fat desktop like KDE, or as leet as you like with a simple tiling window manager.

If you've played around with other Linux distros or a BSD variant and found yourself intrigued on the command line, then you shouldn't find Slackware too challenging. With Slackware, it really pays dividends to spend time with the READMEs before embarking on a serious endeavour such as installing it. As with many OSes, a good way to start experimenting with

Slackware is to install it in a virtual machine using software such as VirtualBox or Qemu.

A great resource for Slackware is **docs.Slackware.com**. At the top of the main page you'll see a link to "Slackware installation" – read that page and you'll get a detailed and clear description of how to install it. We're not going into installation details here, but a crucial point to remember is that it's recommended that you do a full installation. This may seem irksome, especially if you're used to starting with a minimal package set and just adding in what you need, but there's a good reason for it, which we'll come to later. Besides, only 8 GB is required, which even for a five-year-old computer is probably a fraction of your available hard drive space.



Slackware may be over 20 years old, but that doesn't mean users are stuck in the past – here it is running KDE 4 and a bunch of modern applications.

The mythical Slackware package manager

There's a myth that Slackware has no package manager. Like all myths it has some basis in fact, in that Slackware doesn't come with a GUI package manager nor one that automatically handles dependencies, though unofficial tools such as **slapt-get** do exist.

If you are used to Yum, Pacman, **apt-get**, or Ubuntu's GUI tools, we can't blame you for finding this off-putting, but there is a reason behind it. In fact, Slackware users actually grow to appreciate the advantages of handling dependencies manually. And no, it's not a form of spiritually cleansing masochism: Slackware users do not wear horsehair shirts, nor do they birch themselves at daybreak. The reason behind the lack of automated dependency checking is – guess what? – simplicity. Let us explain.

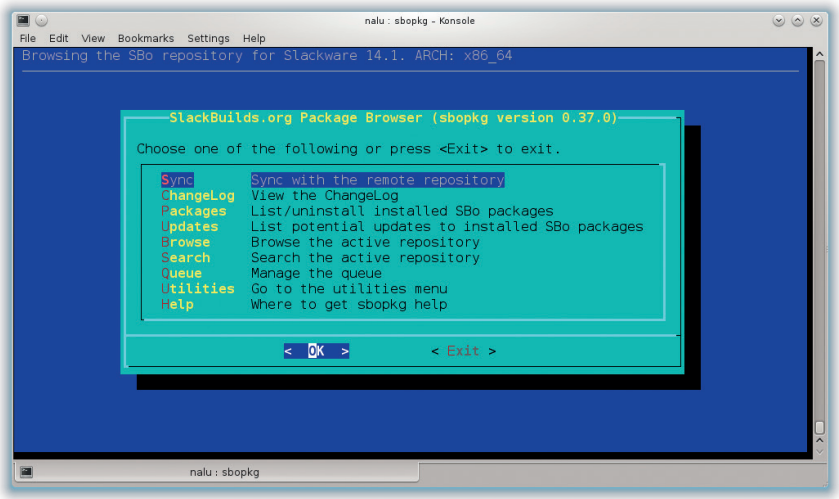
Slackware is designed so that with a full installation you will be provided with tools to address most tasks you'd expect from a modern OS, but even more importantly, most of the libraries you'll need will be installed. So, if you install package 'foo', it's likely that a dependency, say, library 'lib-bar', is present already. Contrast this with Arch, say, where the initial minimal install is likely not to contain 'lib-bar' and Pacman will have to pull in 'lib-bar' and all of its dependencies too. That's no criticism of Arch – in fact Arch is often a loyal Slacker's second distro choice because it offers a different kind of simplicity, one that starts from minimalism.

Tux the penguin is depicted here with J. R. "Bob" Dobbs, founder and prophet of the Church of SubGenius. It's a bit like how a Jedi uses The Force.

I've been using Slackware for nearly 20 years for all kinds of things (scientific computing, audio, software development, gaming) and only once did I have a problem with a package having so many dependencies that it put me off an installation (Pandoc's dependency on Haskell was the culprit). It's not uncommon that there are no dependencies to install, and when there are it's often just one or two. To illustrate the point, these are a few of the packages I have installed that are not included in Slackware 14.1: LibreOffice, Steam, Audacity, VirtualBox, TeamViewer, OwnCloud desktop client and NetBeans. Together they only required three packages to be installed as dependencies (including dependencies of dependencies).

"Slackware users do not wear horsehair shirts, nor do they birch themelves at daybreak."

If you accept that manual dependency resolution in Slackware isn't as big an overhead as you'd expect, then you can start to appreciate the advantage it brings: understanding your system. Here's an example. Recently I got a Dell XPS 13 with Ubuntu 12.04 LTS pre-installed and I grew to like it in many ways in the six months I used it. But, on one horrible morning when I had some urgent work to do, it refused to boot. I ruled out a hardware problem, and after poking around on the recovery console, I guessed that some package I recently installed had trashed the X display drivers. Reinstalling the **Ubuntu-desktop** package and the X drivers seemed to fix it, but many hours were lost. I never really understood what caused the problem – my best guess was that it was either a dependency of Steam or a 3D



sbopkg is an efficient Curses-based tool for browsing and working with scripts from **SlackBuilds.org**.

visualisation package that I had recently installed using the Software Centre.

Now, Slackware is not magic. I have trashed Slackware systems too, but when I did I could usually see the trouble ahead, and could decide to leave well alone until after some important work deadline was behind me. And if I did take the plunge and trash the system I'd know exactly what I'd done and so could fix it fairly easily.

Simple package handling scripts

I rely on a trio of Bash scripts provided with Slackware: **installpkg**, **upgradepkg** and **removepkg**. These scripts do what they say, and take one argument, which can be either, the name of the package, **foo**, or the full filename, **foo-1.6-i686-LV.txz**. There are also a number of options that are explained in the provided man pages, but if you want to know exactly what's going on you can read the scripts yourself with **less /sbin/upgradepkg**.

"Another key difference with Slackware is that upstream sources are rarely patched."

The **installpkg** script extracts the tarball in place in the filesystem tree (**/usr**, **/etc**, **/lib** and so on), and then, if a file called **doinst.sh** exists, it's run for post-installation tasks.

The **upgradepkg** script is simple too – it installs the new version of the package specified, say **foo-1.6**, then checks to see what files were present in the **foo-1.5** package but not in 1.6, and removes them.

If you want to find out what packages you've got installed, all you need do is look in the **/var/log/packages** directory. Every installed package will have a text file present with the same name of the original package file, but without the extension, eg **foo-1.6-i686-LV**. The file will list every file in the package with its full path. In the spirit of being *NIXy, you can then examine the status of packages using file and text commands. For example, if you're wondering if package **foo** is installed and, if so, what version, where from and when you installed it, just do

```
ls -l /var/log/packages *foo*
```

Or perhaps you've found a file such as **/usr/bin/scareyvirustrojon.nasty** and wonder where it's from – this will tell you:

```
grep nastyvirus /var/log/packages/*
```

and hopefully jog your memory into realising it's part of that Humble Bundle game you installed a while ago.

Updated Slackware packages come out soon after upstream maintainers release a security fix. For example, when I read about the infamous Heartbleed bug in a tech news RSS feed, I found there was already an updated package from Patrick Volkerding. When an updated package is released, it's simple enough to download the new tarball and install it with **upgradepkg foo**. However, if a number of updated packages arrive at the same time, it's much simpler to use the **slackpkg** tool instead, which looks much like a command line package manager from other distros, ie:

```
slackpkg update
slackpkg upgrade-all
```

The first line updates the information on the packages, a bit like **apt-get update**, and the second line brings up a Curses menu where you can choose which upgrades to apply. **Slackpkg** is shipped with Slackware and it only works with packages in the official distribution. However, there is a plugin for it, called **slackpkg+**, that enables you to work with a number of other repositories, such as long-time Slackware contributor Eric Hameleers' packages and **slacky.eu**.

Another key difference with Slackware is that upstream sources, including the Linux kernel itself, are rarely patched. Yet again, it's for simplicity's sake. It's simpler for the maintainer (and remember, Slackware really has only one full-time maintainer) but it's also simpler for a user who wishes to alter the system by installing, say, a different version of PHP, or even the kernel itself. It's also a bonus for upstream, because if they get feedback from a Slackware user they can be reasonably confident that the bug wasn't introduced in a patch.

SlackBuilds

Slackers like to know where their packages come from – personally I only install packages from three sources: official; Eric Hameleers' repositories; or those I build myself. A Slackware package is usually created with a SlackBuild script, which is a Bash script that takes the source (or a binary distribution) and turns it into a Slackware package. For example:

```
wget http://SomeTrustedSite.org/foo.tar.gz
tar xvf foo.tar.gz
cd foo
wget http://HomeOfFoo.org/downloads/source/foo-1.6.tar.gz
./foo.SlackBuild
installpkg /tmp/foo
```

First we get a tarball that contains the SlackBuild script and other related files (or we could just write a SlackBuild ourselves). We then extract it and **cd** into its directory, then download the source code. At this point I'd usually stop to glance over the SlackBuild so I can

see what it's going to do. Then the script is executed and the package is installed. Whenever it's available, I also like to check the GPG key or MD5 sum of downloaded files.

At simplest, a SlackBuild script for something that uses the common **autotools** build method could be

```
tar xf foo-1.6.txz
cd foo-1.6
./configure
make
make install DESTDIR=/tmp/foo-1.6
cd /tmp/foo-1.6
makepkg /tmp/foo-1.6-x86_64-LV.tgz
```

The first few lines should be familiar. The fifth line copies the build products (binaries, libraries, icons etc) to the **/tmp/foo-1.6** directory, within which you'll typically find directories such as **/usr/bin**, **/etc**, **/lib/** and so on, corresponding to where the installed files should end up. The final two lines will make the package. **DESTDIR** isn't always supported, but usually there's an equivalent parameter that does the job. Notice that the package file name is conventionally of the form **PACKAGENAME-VERSION-ARCHITECTURE-BUILD**. The last bit is useful to keep track of where a package has come from and who built it.

Usually SlackBuilds have extra lines to standardise them and make them easier to update, eg **$VERSION** could be defined up front, so that the version number **1.6** doesn't have to be scattered throughout the script. Other common additions to scripts are to remove extraneous files left over from the build process, copy README files into the package, add path prefixes such as **/usr**, and even patch the source. The rule of thumb is to stay close to what upstream provided.

The lazy way to slack

I rarely write my own SlackBuild scripts these days – in fact I rarely even run SlackBuild scripts myself. My preferred modus operandi involves two excellent resources from the Slackware community.

The first is **SlackBuilds.org**, often abbreviated to SBO. This is a website with a repository of scripts for packages that are not present in the full Slackware distribution. Each script has its own page with a link for downloading the SlackBuild tarball that includes the SlackBuild script itself with a README and an info file with information about the source, such as download link and MD5 sum. SlackBuilds is an excellent resource and relies on community contributions, but is moderated by a small team who ensure that standards are kept high and that no silly or malicious SlackBuild scripts are admitted – since SlackBuilds are often run as root, you really don't want one to have **rm -rf /** lurking inside it. It's also worth noting that all the scripts on **SlackBuilds.org** will assume a full Slackware installation.

The second essential resource is **sbopkg**, which is a Curses-based tool that makes it easier to work with **SlackBuilds.org**. It allows you to search for a package by any fragment of its name, and once you've found it, you can peruse the concise README and then check to see what dependencies it has. If there are none, then a key press tells **sbopkg** to download the source, check MD5 sums, run the SlackBuild and install the package. If there are dependencies, you can add the current package to a queue for later processing, and then search for each dependency, adding them to the queue as you go. Once you've queued them all up, hit a key and it'll process the entire queue.

If hunting down dependencies really does become a chore, you can use a tool called **sqg** to generate queue files which effectively turns sbopkg into a package manager with automated dependency resolution.

The future?

Slackware has been around for a while, and it shows no sign of going away anytime soon – but what if Patrick Volkerding decided to abandon it, or he became unable to continue? Most of the main distros aren't reliant on one person, and even if, say, Red Hat Inc went under, it's probable that Fedora would continue, just as Mageia was forked from Mandriva after its company got into trouble. It's likely that Slackware too would continue in some form with volunteers from the community stepping up, but the distro would inevitably change in character because Slackware is imprinted with Patrick's personality.

I've spent serious time with Ubuntu, CentOS, Mint, Red Hat, Arch and Tiny Core, and learned different things from each of them, and can see how they suit others' needs, just as Slackware suits mine. Slackware is like a very cleverly designed machine in which you can see how all the various intricate parts work together. Not only can you gain a deep insight into GNU/Linux by spending time with Slackware, but you will feel in control of your software and hardware. In an era in which we have good reason to believe that our consumer electronics, in sealed units with proprietary software, may be used to spy on us, this is especially reassuring.

Old yet up-to-date

As Linux emerged from infancy, Slackware became the most popular distribution by proving its worth as a more reliable fork of the now defunct Soft Landing System (SLS) distro. Debian was released slightly later in 1993 and, in contrast to Slackware, evolved with a democratic and distributed philosophy, and now has an impressively large community and has spawned many more derivatives, most notably Ubuntu. There are quite a few distros based on Slackware, including Slax, Vector Linux Zenwalk and Salix – each of them adding different things, such as a live CD/USB version and more featureful (but more complicated) package managers.

Slackware has just one professional maintainer, Patrick Volkerding, who draws his living from subscriptions, merchandise and donations. It does have a strong community, which can be found on IRC, but I generally seek help and offer advice on the forums at **LinuxQuestions.org**. A few community members are worthy of mention, in particular Eric Hameleers (aka Alien Bob) and Robby Workman, who you might call core contributors to Slackware, and Stuart Winter (aka drmozes) who maintains the ARM port.

Slackware is on version 14.1, released in November 2013 with the 3.10.17 kernel in 32-bit, 64-bit and ARM editions, and UEFI hardware installation is supported. There's also an elegant multi-lib (ie supporting both 32- and 64-bit) solution maintained by Eric Hameleers. Releases have averaged at about once per year recently, but if you feel a bit more bleeding edge, you can keep updated to Slackware-current which will eventually become the next release.