

# LINUX 101: MASTER YOUR PACKAGE MANAGEMENT SYSTEM

MIKE SAUNDERS

apt-get, dpkg, yum, zypper... There are many ways to install packages on your Linux box. Here's everything you need to know.

**WHY DO THIS?**

- Understand how packages work and what exactly they provide
- Learn vital admin skills to manage packages outside of the GUI
- Discover how packaging systems work across other distros

Package management systems are both loved and hated in the Linux world. On the one hand, they provide efficient ways to install and remove software, with everything neatly bundled up. (Contrast this to Windows, where a **setup.exe** typically scatters all sorts of stuff all over your hard drive and registry, and running the “uninstaller” doesn't get rid of everything. You can even find third-party “uninstall” tools designed to clean up this hideous mess.)

On the other hand, package management systems often make it difficult to get the latest hot new applications. You have to find the right repository for your distribution, and make sure dependencies are satisfied (usually this is automatic, but not always), and so forth. And if you're completely new to Linux, you might find all of the terminology here baffling. So in this tutorial we'll explore the two main packaging systems used in GNU/Linux distributions, and provide some advanced tips and tricks for long-time Linuxers as well.

**Dissecting the jargon**

First of all, let's clear up any confusion by defining some terms:

- **Package** A single, compressed file that contains a program or related files such as a supporting code library, documentation, artwork or video game level data. Some (usually small) programs are provided in single packages, whereas larger application suites like KDE and LibreOffice are supplied in multiple packages (to make updates easier, as you don't have to download the whole lot each time).
- **Dependency** Every package includes some metadata, such as other packages it depends on. For instance, the AbiWord word processor uses the GTK toolkit for its interface – a library that is supplied separately – so the AbiWord package will

```
mike@debianmike: ~
File Edit Tabs Help
root@debianmike:/var/cache/apt/archives# dpkg -I vim_2%3a7.3.547-7_1%386.deb
new debian package, version 2.0.
size 778220 bytes: control archive=1871 bytes.
 934 bytes, 22 lines control
 237 bytes, 4 lines md5sums
 2395 bytes, 77 lines * postinst      #!/bin/sh
 1218 bytes, 57 lines * preinst       #!/bin/sh
Package: vim
Version: 2:7.3.547-7
Architecture: i386
Maintainer: Debian Vim Maintainers <pkg-vim-maintainers@lists.aliases.debian.org>
Installed-Size: 1797
Depends: vim-common (= 2:7.3.547-7), vim-runtime (= 2:7.3.547-7), libperl (>= 2.2.0-8), libc6 (= 2.11), libgpm2 (= 1.20.4), libssl1.0 (= 1.0.2), libtinfo5
Suggests: ctags, vim-doc, vim-scripts
Provides: editor
Section: editors
Priority: optional
Homepage: http://www.vim.org/
Description: Vi IMproved - enhanced vi editor
 Vim is an almost compatible version of the UNIX editor Vi.
 Many new features have been added: multi level undo, syntax highlighting, command line history, on-line help, filename completion, block operations, folding, Unicode support, etc.
 This package contains a version of vim compiled with a rather standard set of features. This package does not provide a GUI
```

Here's the metadata for the Debian Vim package, obtained with the **dpkg -I** command. Note the highlighted line, showing dependencies.

list GTK as a dependency in its metadata. Package systems normally handle dependencies automatically, although it can get messy.

- **Repository** An online store for packages. Most Linux distributions have their own repositories (or “repos”) with up to tens of thousands of packages. Some software developers make their own third-party repositories that can be used alongside the official distro ones.

These terms, and the general workings of packaging systems, apply across almost every Linux distribution. There are some technical differences in the implementation of packaging systems, and command names vary, but the underlying principles are the same.

Most desktop-focused distros include graphical package managers; in this tutorial, however, we'll focus on the command line tools, as they're usually much more versatile and teach you a lot more about what's going on.

**1 DEBIAN/UBUNTU: APT AND DPKG**

Let's start with the system used by Debian, Ubuntu and other distros based on these two. Apt (which stands for the “Advanced Packaging Tool”) provides a suite of utilities for locating, downloading and managing dependencies of packages.

The **apt-get** tool installs a program. For instance, say we want AbiWord; open a terminal and enter:

```
sudo apt-get install abiword
```

(This needs to be run as root, the administrator user, hence the **sudo** command at the start. On Ubuntu-based distros you'll be asked for your user account password. If you're on Debian, the command is **su -c “apt-get install abiword”** – modify the rest of the **sudo** commands in this tutorial to use **su -c** with quotes instead. You'll be asked for the root password in this case.)

Before downloading AbiWord, Apt will tell you which dependencies it's going to retrieve, show you how much drive space is going to be used, and check for confirmation. Hit Enter to go ahead, or N to stop. Apt will pull the packages from the internet repositories and install them.

Now, that **apt-get** command is great when you know exactly what you're looking for – but what if you don't know the name of a package? Try this:

#### apt-cache search "word processor"

Aha! This lists all packages in the distro's database that have "word processor" in their descriptions. (If it's a long list, pipe it into the **less** text viewer, like so:

**apt-cache search "word processor" | less**. Hit Q to quit the viewer.) Note that we don't need **sudo** in this case, because merely searching the database isn't an administrative command that changes system files.

The next question you're probably asking is: how does Apt retrieve and store all of this information? Every time you do this command:

#### sudo apt-get update

Apt retrieves the latest package information from the repositories, and stores the details in **/var/lib/dpkg**. (Also note that Apt caches packages in **/var/cache/apt** after downloading, which can take up a lot of space, so use **sudo apt-get clean** to remove them.)

Note that this command merely updates the database, and doesn't actually update your system to the latest version of the packages. For that you need to enter:

#### sudo apt-get upgrade

### Begone, unwanted apps

There are various ways to remove a program, which may seem a bit odd at first, but when you compare it with the aforementioned mess on Windows it makes a lot of sense. First the simplest way:

#### sudo apt-get remove abiword

This gets rid of the program, but not any system-wide configuration files. (This isn't a big deal with a desktop program, but imagine if you've spent hours configuring a mail server, and need to remove it

### Advanced tip: Converting RPMs to Debs

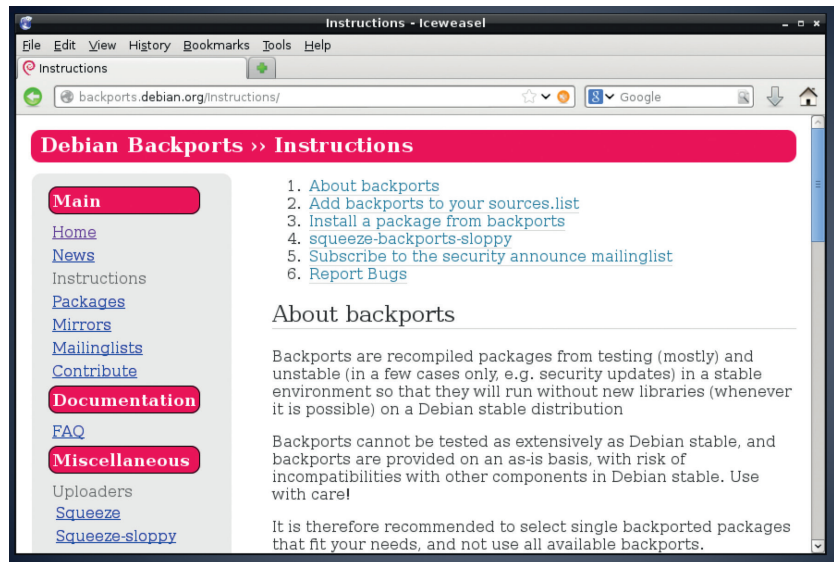
It should be an absolute last resort, but if you really need it you can use a tool called Alien to convert RPM packages to Deb files and vice-versa. However, due to the technical and implementation differences between these package formats, along with the usual plethora of different file locations and library versions across distros, the results are rarely pretty. To use it (as root):

#### apt-get install alien

#### alien --to-deb <filename.rpm>

(Use **--to-rpm** if converting the other way round.) If you want pre- and post-installation scripts to also be transferred into the new package, add the **--scripts** option.

For small, single-package programs with limited (or statically compiled) dependencies, Alien can sometimes be a life-saver when you have no other options. But it's a bit of a hack job, and shoehorning one distro's package into another distro usually results in a broken app. Beware!



temporarily for some reason. If the **apt-get remove** command also deleted your hand-crafted config file, you'd be gutted.) So to remove all configuration files:

#### sudo apt-get remove --purge abiword

This has totally removed the program from the system, but its dependencies still remain. If you want to remove those as well (providing that they're not being used by any other program) then follow up the previous command with:

#### sudo apt-get autoremove

### dpkg

There's also a more low-level **dpkg** utility, which handles the nitty-gritty of installing and removing packages. Here are some of its more useful commands:

- **dpkg -l** lists all installed packages. You can show the details (version number and short description) for a single package with **dpkg -l abiword**.
- **dpkg -i <package.deb>** this installs some package (for example, **package.deb**) that you have downloaded. It's a useful command if you've got a program off a website, although repositories are the better method.
- **dpkg -L abiword** lists all files inside the package.
- **dpkg -S /path/to/file** this shows which package contains **/path/to/file**. So **dpkg -S /bin/ls** shows that it's part of the **coreutils** package.

### Adding repositories

Debian-based distributions store their repository information in **/etc/apt/sources.list**. This is a plain text file containing URLs from which packages can be retrieved, along with the codename of the distribution (eg "wheezy" for Debian 7) and the types of packages (eg "main" for free/open source software from the main Debian developers, "non-free" for packages that have licence issues etc.) You can add repositories to that list as you discover them on the web – just remember to do **apt-get update** afterwards so that your local database is in sync.

At <http://backports.debian.org> you'll find repositories that provide up-to-date applications for older Debian stable releases.

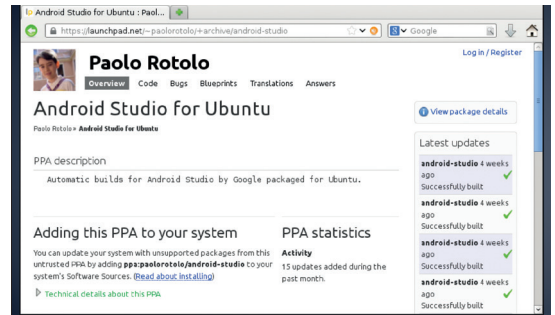
### LV PRO TIP

To extract a Deb file by hand, run **ar x <filename.deb>**. This creates three files in the current directory: **data.tar.gz** (containing the program's files, typically extracted into **/usr**); **control.tar.gz** (containing the package's metadata, such as dependencies); and **debian-binary** (the version of the **.deb** file format being used, usually 2.0). Sometimes the control and data files have different compression formats, and end in **.bz2** or **.xz**.

If you plan to add multiple repositories from different sources, it's better to place them in separate files in the `/etc/apt/sources.list.d` directory. This makes them easier to manage and remove, and means your distro can manage the main `sources.list` file without getting confused by your modifications.

If you're using Ubuntu or Mint, you'll often come across PPAs (Personal Package Archives). These are repositories set up by developers and third-party users to provide packages that aren't officially in the distribution – or newer versions of packages. Most flavours of Ubuntu and Mint only receive package updates for security holes or bugfixes, and you have to upgrade to a new version of the distribution every six months if you want the latest software – not always an ideal situation. With a PPA, you can get new versions of software for your existing distribution, without having to wait or upgrade, so they're very popular among users who want to live life on the bleeding edge.

A PPA typically includes the name of the developer along with the name of the program, so here's an example: Paulo Rotolo has packaged up Android



Many PPAs are available for Ubuntu and Mint, providing packages that aren't officially part of the distros.

Studio for recent versions of Ubuntu. On his page at <https://launchpad.net/~paolorotolo/+archive/android-studio> you'll see that his PPD is called `ppa:paolorotolo/android-studio`. To install the program you'd enter the following:

```
sudo apt-add-repository ppa:paolorotolo/android-studio
sudo apt-get update
sudo apt-get install android-studio
```

You'll find many PPAs on the web, and they're a great way to try new apps quickly.

## 2 RED HAT, FEDORA, OPENSUSE: YUM, ZYPPE, URPMI

Let's move on to the RPM-based distros. RPM was originally the "Red Hat Package Manager", due to its origins in that distro, but today it's known as the "RPM Package Manager" (yes, a recursive acronym) due to its use in many other distros. Unfortunately, things get a bit fragmented here, with each RPM-based distro using its own toolset. Most of the commands are similar though.

Fedora and Red Hat Enterprise Linux use the Yum package manager for searching and downloading packages, while the `rpm` command does the work of installing. To find a program, do:

```
yum search abiword
```

And to install (switch to root with `su` first):

```
yum install abiword
```

Removing packages is easy (`yum remove <package>`), as is updating the distribution to the

latest packages in the repositories (`yum update`). To get rid of unused dependencies that were installed by programs you've since removed, use `yum autoremove`. And to remove cached packages after a big download, enter `yum clean packages`.

You can get detailed information about a package, such as whether it's installed or not, like so:

```
yum info abiword
```

And to see which dependencies a package has, try `yum deplist <package>`. To generate a complete list of all installed packages, enter `yum list installed`.

Yum stores its repositories in plain text files in the `/etc/yum.repos.d` directory; to add a new repository, use this command:

```
yum-config-manager --add-repo <URL>
```

(Simply delete the file in `/etc/yum.repos.d` to remove the repository.)

### RPM

Yum is the tool you'll want to use most of the time, but for more low-level work involving individual packages that you've downloaded, there's the `rpm` command. For instance, `rpm -qpi <package.rpm>` displays information about a locally stored package (`qpi` stands for 'query package information'), and `rpm -i <package.rpm>` installs it.

You can also use `rpm` to find out which package a file belongs to:

```
rpm -qf /path/to/file
```

And to list the contents of a package, use `rpm -ql <package>`.

Unusually, RPM uses the `cpio` archive format for its packages – a format that few people have heard of.

### Advanced tip: CheckInstall

In LV005 we looked at compiling programs from their source code (p86). You may recall that the `make install` step places the program's files in your filesystem – usually in subdirectories of `/usr` or `/usr/local`. Wouldn't it be better, though, if you could bundle up the newly installed files into a package, for easy distribution and removal?

Well, you could learn the highly complicated art of making packages by hand, or use `CheckInstall` instead (it's provided in most distros' repositories). This monitors all files created in a `make install` operation and generates and installs a package

accordingly. So instead of entering `sudo make install`, you'd enter `sudo checkinstall`.

If we do that using Alpine (the example app that we compiled last issue), we end up with a package called `alpine_2.11-1_i386.deb`, and `CheckInstall` has also installed it. Now we can easily copy that package to another machine (as long as it's running the exact same distro!) and remove it using the commands mentioned earlier in this guide.

Note: packages generated by `CheckInstall` are very specific to your own distro setup, and lack proper meta data information, so they may not work elsewhere.

Consequently, it can be difficult to remember the options used to extract files. If you need to extract a .rpm file, first move it into a separate directory (to stop it potentially overwriting files in the current one), and then enter this command:

```
rpm2cpio <package.rpm> | cpio -idmv
```

Of course, you should replace <package.rpm> here with the real package filename. This creates a directory structure in the current directory that would normally be extracted into the root (/) directory when installing the package.

It's worth noting that Yum will be around for a few more Fedora releases, but ultimately the goal of the distro developers is to move to a new package manager, DNF, which forked from Yum in 2012. DNF should be largely compatible with Yum, so most of the commands will be identical or similar, and in Fedora 22 entering yum will actually run dnf and display a warning message.

### OpenSUSE and Mageia

Let's look at the most common commands for these distributions. OpenSUSE and Mageia are also RPM-based distros, so they have the rpm tool available and it works in the same way as in Fedora, but they have their own higher-level package management tools. OpenSUSE uses Zypper, while Mageia (the Mandriva spin-off) has Urpmi. The following commands show how to:

- 1 Find a package or program
- 2 Install a package
- 3 Remove a package
- 4 Update the package database
- 5 Update the system
- 6 Add a repository
- 7 Get information on a package

First in OpenSUSE:

1. zypper search <package>
2. zypper install <package>
3. zypper remove <package>
4. zypper refresh
5. zypper update
6. zypper ar <URL> <alias>
7. zypper info <package>

And then for Mageia:

1. urpmf --summary <search word>
2. urpmi <package>
3. urpme <package>
4. urpmi.update -a
5. urpmi --auto-select
6. urpmi.addmedia <name> <URL>
7. urpmq -i <package>

The urpmf command is especially useful if you're missing a dependency, and you need to find out which package has it. For instance, if you're trying to compile a program and the build script complains that the foobar.h header file is missing, you can do urpmf foobar.h and find out which package contains it.

So, those are the major Linux package managers covered – now you should be able to jump between

**Zypper Cheat Sheet**

More Information: [https://en.opensuse.org/SDB:Zypper\\_usage](https://en.opensuse.org/SDB:Zypper_usage) or type `man zypper` on a terminal

For Zypper version 1.0.9

**Basic Help**

zypper #list the available global options and commands  
zypper help [command] #Print help for a specific command  
zypper shell or zypper sh #Open a zypper shell session

**Repository Management**

**Listing Defined Repositories**

```
zypper repos or zypper lr
```

Examples:  
zypper lr -u #include repo URI on the table  
zypper lr -P #include repo priority and sort by it

**Refreshing Repositories**

```
zypper refresh or zypper ref
```

Examples:  
zypper ref packman main #specify repos to be updated  
zypper ref -f upd #force update of repo 'upd'

**Modifying Repositories**

```
zypper modifyrepo or zypper mr
```

Examples:  
zypper mr -d 6 #disable repo #6  
zypper mr -rk -p 70 upd #enable autorefresh and rpm files 'caching' for 'upd' repo and set its priority to 70  
zypper mr -ka #disable rpm files caching for all repos  
zypper mr -ki #enable rpm files caching for remote repos

**Adding Repositories**

```
zypper addrepo or zypper ar #followed by the repo url and alias
```

Example:

**Package Management**

**Selecting Packages**

By capability name:  
zypper in perl(Log-Log4perl)  
zypper in qt

By capability name and/or architecture and/or version  
zypper in zypper-0.12.10  
zypper in zypper.i586-0.12.11

By exact package name (--name)  
zypper in -n ftp

By exact package name and repository (implies --name)  
zypper in factory:zypper

By package name using wildcards  
zypper in yaast\*ip\*

By specifying a rpm file to install  
zypper in skype-2.0.0.72-suse.i586.rpm

**Installing Packages**

```
zypper install or zypper in
```

Examples:  
zypper install git

By capability they provide  
zypper in MozillaFirefox l3 3

Others:  
zypper in yaast #install all yaast modules  
zypper in -t pattern lamp\_server #install lamp\_server pattern (packages needed for a LAMP server)  
zypper in vim-emacs #install vim and remove emacs  
zypper in amarok updt:libxine1 #install libxine1 from updt

**Removing Packages**

```
zypper remove or zypper rm
```

Examples:  
zypper remove sqtite

**Source Packages and Build Dependencies**

```
zypper source-install or zypper si
```

Examples:  
zypper si zypper  
Install only the source package  
zypper in -d zypper  
Install only the build dependencies  
zypper in -d zypper

**Updating Packages**

```
zypper update or zypper up
```

Examples:  
zypper up #update all installed packages with newer version as far as possible  
zypper up libzypp zypper #update libzypp and zypper  
zypper in sqtite3 #update sqtite3 or install if not yet installed

**Zypper in Scripts and Applications**

**Non Interactive Mode**

```
zypper --non-interactive
```

Examples:  
zypper --non-interactive patch #skips all interactive patches which would require user confirmation

**No GPG Checks Mode**

```
zypper --no-gpg-checks
```

**Auto-agree with Licenses**

```
zypper --auto-agree-with-licenses
```

See <http://en.opensuse.org/images/1/17/Zypper-cheat-sheet-1.pdf> for a handy Zypper cheat sheet (<http://tinyurl.com/a7dbnl6> for the second page).

distros more easily, and you know more about what's going on under the hood.

### Other packaging systems

While Deb and RPM dominate the Linux world, some distros have their own packaging systems that are worth knowing about. Arch Linux, for instance, sports the Pacman system, which is very highly regarded among its users. Arch can be a challenging distribution to maintain, due to the fact that it's constantly changing, but Pacman handles the task of upgrading with aplomb.

Slackware, meanwhile, is famed for being one of the most traditional Linux distros (and it's also the longest-running Linux flavour in existence). It's often criticised for not having a package manager, but that's not entirely fair, as we explore on page 26.

While most packaging systems take the approach of extracting data into /usr, and perhaps with some bits and bobs in /etc and /var, there are some more ambitious systems that attempt to make things simpler. In the Gobo Linux distribution, for example, all applications are installed in the /Programs directory, and you can have multiple versions of the same application. So you could have /Programs/LibreOffice, and inside that directory you'd have subdirectories for 4.1, 4.2 and so forth. This keeps programs neatly separated from one another, and makes it easy to install and delete them – you don't need the package manager to do a lot of black magic.

Shared libraries are a potential problem here, but Gobo Linux works by using symbolic links in the /System/Index/lib directory. So you might have GTK installed in /Programs/GTK+/3.0, and programs that use it won't necessarily know that it's there. But they will find libgtk.so in /System/Index/lib.

Mike Saunders has been installing, removing, creating and breaking packages for 15 years. There's no stopping him!