Installing applications and packages

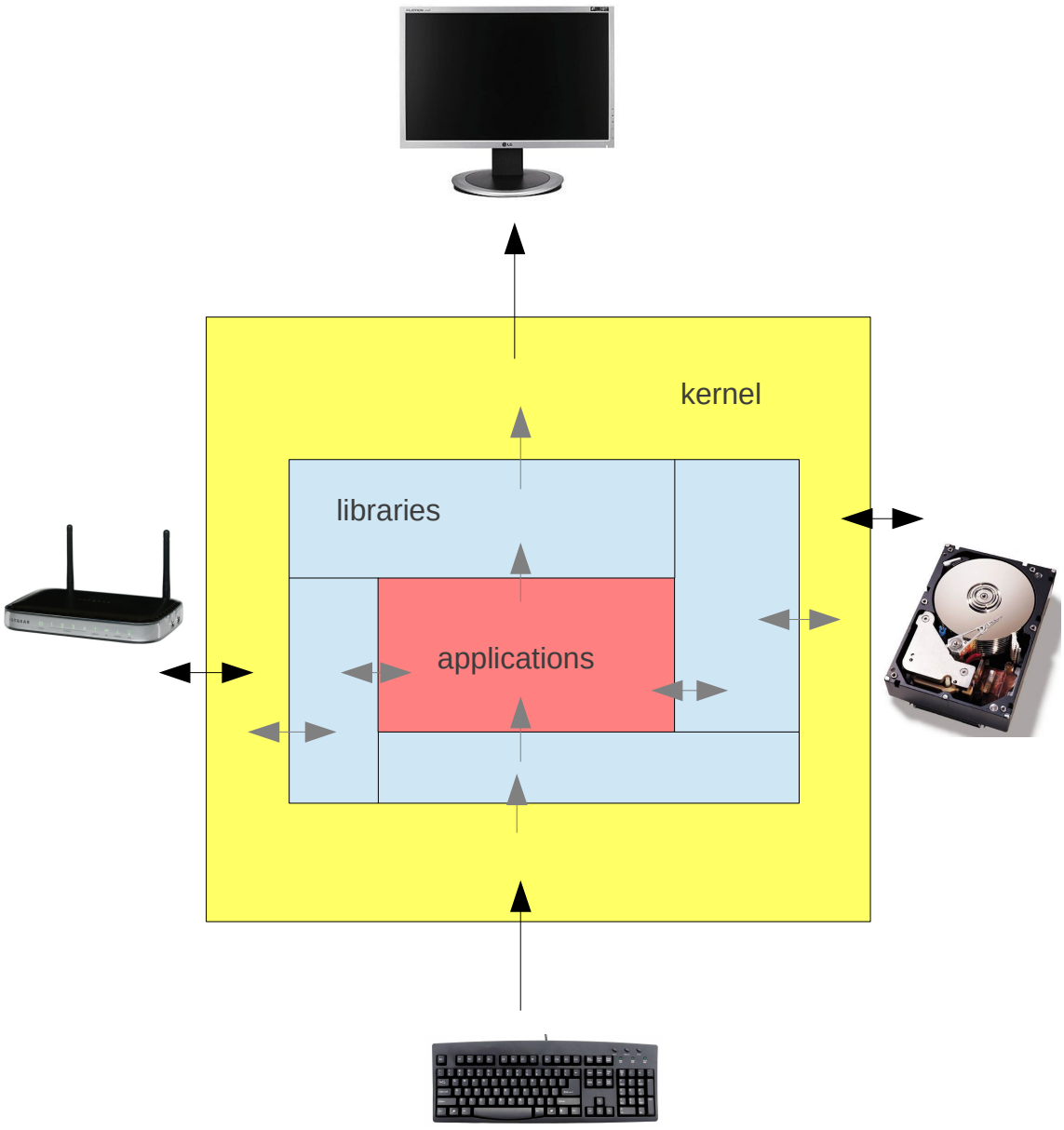
Andy Pepperdine

This document describes how to install applications on a Linux system, and what packages are. It will give some background to explain some of the thinking behind the way in which applications are distributed and why.

It is assumed that a Linux distribution based on Debian is in use (such as Ubuntu or Mint) as that is the most common these days.

Applications, libraries and the system

The following diagram shows a representation of how parts of the system fit together.



You think you are interacting with an application, whereas in reality you interact with the kernel which passes the requests to a library, which in turn passes it on to the application, which then interprets it as some action to take, and sends the answer back along a similar path. The kernel controls all of the external devices – the keyboard, touch pad, mouse, screen, internet connection,printer, disks, USB sticks, microphones, webcams, etc. Nothing happens without the kernel seeing and controlling it.

The important thing to note here is that an application relies on libraries and the kernel for some of its work. It does not contain the libraries, since they can be and are used by several applications at the same time; for example, both a web browser and an e-mail client may access the internet simultaneously. So it makes sense to separate those functions out, giving you just one way of doing it with consistent results for all applications.

In practice, it is a little more complicated. For example, libraries will use other libraries for part of their work, etc.

Distributions, repositories, applications and packages

If you run a distribution, say Ubuntu or Mint, then the distro provider has selected which libraries and applications they will support, and prepared packages containing each of them. They will also have recorded what each application relies on in the way of libraries. The libraries will be packed into their own packages, with their own list of required other libraries, and so on.

These packages are then placed into repositories. Each distro will have its own set of repositories, and they will ensure that the applications and libraries are compatible with each other. It is not in general possible to mix repositories from different distros as they may not be compatible.

Your system will periodically check for updates to the applications installed on your system. It only knows about those applications installed by synaptic, or similar means, and will check all the repositories it has been told about for any updates to the applications that are installed. Any applications that are not in a repository cannot be checked.

Repositories

Each Linux system from a distro that uses repositories maintains within itself a database recording all the software that is installed, and what is available on the supported repositories of the distro. This database tells us what the version of the software is that is currently installed.

If for some reason, a new version is produced by the distro (for example to fix a security hole) then they will place it on the repository. Your system can then periodically, or on demand, interrogate the repositories and determine the differences between the version numbers there and the versions installed on the system. If there is an installed application with a later version in the repository, then you can be alerted to the update. At this point, you can then install, if you wish, the newer version.

The great advantage of using repositories, is that you can keep up to date automatically with any security changes, which the distros will assiduously update as soon as they can.

Installing an application

An application cannot be used until. it is on your PC, and all the things it needs are also there. If it is the repository of the distro, then it is merely necessary to go to the software centre (Ubuntu) or start the application called *synaptic* (all Debian based distros), select the application you want, and ask it to be installed. The system will note which other libraries and packages are required for the request, and collect those as well. All of the requested packages will then be fetched from the repositories across the internet and then processed by placing them in their expected locations in the system, adding it into any menus, preparing other data areas, and all sorts of other things it might need in order to work. This is the process known as installing.

What if it is not in the repository?

There are various other places you may get an application from, and they have different characteristics, depending on how it has been prepared for you to download. We will look at these in turn.

Packaged by the developers

Sometimes a distro has not prepared a package, but the developers of the application have done some work already to provide something in the right format for some distros. Typically, this may be to build their own package in the correct format for some distro (typically Debian, Mint, Ubuntu etc.) and support their own repository to contain it. These repositories are known as Personal Package Archives (PPAs). It is then necessary to tell synaptic that there is another repository you want it to keep track of. The owners of the repository will tell you on their website how to do that, and where their PPA is held. Typically, the instructions are quite clear.

An advantage of using a PPA for an application is that the PPA is treated like any other repository. In particular, it will check the PPA for any updates

For example, LibreOffice support their own repository for Ubuntu-like systems, and the announcement for their latest 3.6 release is here: https://launchpad.net/~libreoffice/+archive/libreoffice-prereleases

But perhaps you ought not to go to the pre-releases for anything important. Search for the last stable version, for example here: http://www.upubuntu.com/2012/08/install-libreoffice-36-final-from-ppa.html

But one thing to be aware of, if you step outside what the distro itself provides, is that you are relying on the developers being able to match any changes the distro may make and keep the repository compatible with the official distro.

In practice, I have not noticed any problems doing this, but there is always a small risk.

Not in a package, but in a .deb file

Some applications do not appear in a package at all. You will need to install these in some other way.

The easiest of this type have been produced as a *.deb* file. These files are the basis of packages in repositories, so are compatible with them. However, since they are not in a repository or PPA, you will not be getting automatic updates when they change. You would have to keep track of them yourself and re-install for each later release.

They do however allow your system to ensure that all your existing packages are compatible with this application, and will warn when you try to install something else that is not compatible with it, although it cannot fix the problem – that is up to you.

The simplest way to install from a *.deb* file is to use the tool **gdebi**, which will be found as a normal package in your distros repositories. After installing gdebi, using the file browser to locate the *.deb* file and clicking on it should start gdebi and it will install. gdebi will also download and install any other packages that this *.deb* file requires. In the past, I've had trouble if there are many interdependent packages to be installed together, and in that case, the command should be resorted to.

Some applications (for example, LibreOffice) contain many packages in themselves. These are available from their website as compressed archives of all the .deb files together, and must first be unpacked. The file type will be something like *.tar.gz*, and must be first unpacked to allow access to the contents. A typical sequence is to open the file with the Archive Manager and unpack it. Then open a command terminal and change the working directory to the directory that was created when unpacking the archive. To install all of them at once, do:

```
sudo dpkg -i *.deb
```

However, it would be usually advisable to remove the existing version of the application from your system before adding the new one. It is not normally necessary, but it would ensure that the new version is cleanly installed.

Other methods

In more obscure cases, there is not even a tie-up with any packages, and you probably only have what is known *source files*.

When an application is being developed, the developer will be editing the source files. This is the human readable form of the application. From these files, the application is built by a process known as compiling and linking. Modern programming techniques use other advanced tools to control the whole process of constructing an application. To distribute su8ch a thing to as wide a group of people as possible, typically, the developer will combine all the source files into a single archive file, known as a tar ball, which will contain in compressed form all the source text for the

application, including the data to show how to build it. These collections are called **tar-balls**. But it will not contain the tools to build it, nor any other libraries it will need to execute correctly. The file type for such an archive is probably *.tar.gz* (or equivalently, *.tgz*) or *.tar.bz2* (or *.tbz2*). The *.gz* or *.bz2* indicates the type of compression used, and *.tar* indicates that the file has been created using the Unix *tar* program. [History; *tar* is an abbreviated form of “tape archive”, which tells you how old this format is.]

Modern systems have a tar program that can both uncompress and expand the archive into its component parts. You can probably find the options to use from the site you got the application. For example, if the file is of type *.tar.gz*, then you should create a suitable empty directory to act as the working area, and place the file in it. Then change your working directory to it, and issue the command

```
tar xzvf file.tar.gz
```

It will unravel it all and you should find one or more directories created as a result. If there is a **readme.txt** file, read it, and note the contents – it may contain something useful.

If all goes well, then all you will have to do is change the working directory to the right place, and issue the commands:

```
configure
make
make install
```

You may need to run the install as the administrator, depending on whether the application will be installed for anyone to use, or just a single user.

Needless to say, if you are doing this low level stuff, things can go wrong, and almost certainly will the first time you try. The configure step will check that all the tools you need to build the program are in place on your system. It will report if anything is missing. If you've never done any of this before, then it is certain that there will be something missing. When you know what, you will have to find the package that contains the relevant tool and install it, and try again.

A further complication is in the libraries. You may think that you have the libraries required, and indeed you may well have them installed suitable for the application to run. But that is not good enough to build them. You also need a description of how the library can be used. These packages usually have the suffix **-dev** added to the package name. In order to compile a program to use the package, the *-dev* package will be needed.

If you have ever compiled a program from one of these tar balls you will appreciate the package structure that has already done all the work for you in ensuring that the requirements for each package are known and automatically installed as well, because you have to discover all that for yourself with a tarball. My experience is that the developers try to list the dependences in the readme file, but often miss something out.

Other packaging systems

The *.deb* file type defines the format for repositories built for Debian-based distros. There are others, of which the most popular is *.rpm* for packages based on the RedHat package Management system. For those distros (Suse, Fedora, etc.) there are applications very similar to synaptic for installing and updating the system from the repositories that the distro maintains.

Although there are ways of converting between these types so that, for example, you can load a *.rpm* package onto a Mint system, it is not recommended as they may not be compatible. It is far preferable to ferret out a suitable Debian friendly package instead. Having said that, whenever I have used this in the (distant) past, I never had trouble with it, providing I remembered that dependences were not necessarily properly recorded.

For further information see the Ubuntu pages: http://manpages.ubuntu.com/manpages/gutsy/man1/alien.1p.html

Drivers

Drivers are pieces of program that interact with and control the hardware attached to your machine. Since they are operating at the lowest level of hardware, they are used by the Linux kernel to perform the actions that the kernel requires of the hardware. By far the easiest hardware to use is what the kernel already has support for built-in, and these are usually generic drivers covering a range of types of equipment and makers.

However in some cases, there is no generic driver, and then we have to rely on code provided by the maker of the device. Under Linux, I know of at least two different types of driver, special for Linux, depending on where they execute their code. In addition, there is wrapper that may be used for others drivers that have not been converted from Windows implementations and may work in some cases.

Because of these variations, it is difficult to apply common methods of installing an extra driver if the kernel does not recognise a device. In general, there is a lot of help available if you search for it, but it is essential that you obtain the **exact name and model number** of the device. Manufacturers have a habit of changing the electronics significantly inside their products even when moving from one model number to the next in line. There have even been cases where they have not changed the model number when changing the insides, and so in that case you will need the number of the internal electronic chips. But that sort of case is thankfully rare and some patience will be needed to resolve it.

Here are some useful methods of getting the relevant information. First, there is a graphical interface to list the hardware on your system. called **hardinfo**. If it does not exist, you should install it as it can also list some other useful information about you system. It appears that it does not necessarily integrate into the menu system, in which case open a terminal and issue the command:

```
hardinfo &
```

The trailing & allows you to continue to use the terminal after it has started, otherwise the terminal window would be locked until you close the window.

There are three command line commands that can help identify the devices you have. These are

```
sudo lspci
sudo lsusb
sudo lshw
```

which push out lots of textual information that you have to search through to get the detail of what you need. They can be used by a normal user, but to get all the information you must execute them under the administrator's account.

Maintaining applications

To ensure that you get all the updates to fix severe problems (like security issues), then you should ensure you keep your applications up to date with respect to the distro's repositories, which is normally done in some automatic way by the Update Manager the distro provides.

Very occasionally, it may happen that you do not update for a significant amount of time. In that case it is possible that the accumulated updates become incompatible with the packages and / or repositories you have in your system. When that happens, cancel the update, and use Synaptic. Reload all the repository details and upgrade a few at a time. If you have usual luck, then this will sort it all out with no problems, but if not, then you will find it will block some upgrade. Ignore that one for now, and do the rest, then go back and upgrade the troublemaker.

To reload the repository details, use the Reload icon.

To find what can be upgraded use the Status button to gie a list of the status types of all packages. If any can be updated, one of the lists will be marked Installed (upgradeable). Clicking on that will show the list of what can be done. Then select all those you want to upgrade in one go. A right click will bring up a menu to enable you to select the upgrade. Then the Apply icon will do the job.