MatrixDB - Theory and Practice

Josef Spillner

24th August 2002

Chapter 1

Introduction

A key element for Free Software, and more than often subject for discussion, is package management. A magnitude of ideas has been implemented in recent years. MatrixDB has initially been written to handle complex abstract package selections, but always with generality in mind, so most of the concepts can be applied to similar efforts. In effect, I don't think MatrixDB is something very new - it is merely a result of the previous efforts.

Chapter 2

How to manage packages

There are several ways of getting the software one likes onto one's hard disk, and to keep it up to date. Among all of those choices, some popular ones will be presented.

2.1 Source archives

2.1.1 Single packages

This is obviously the oldest method. People fetch some source, which is nowadays mostly shipped in .tar.gz or .tar.bz2 format, unpack it, compile and install it. This is a very flexible method, but it requires the user to be familiar with both compile-time and run-time dependencies. Most power-users or developers thus hardly use this method to setup a whole system, but profit from the unbeaten up-to-date status for individual packages.

2.1.2 Managed source trees

Distribution methods of source code which takes care of dependencies and the user's preferences have been very popular especially within the BSDs. The so-called Ports Tree keeps a list of available packages, which can be installed anytime, and be removed on request. Linux distributions favoring this way include Slackware (providing a sink for many source packages) and Gentoo (using the Portage system similar to BSD ports).

2.2 Binary packages

2.2.1 Single packages

This is something which the majority of users should be familiar with: Software which is shipping pre-compiled for a certain number of architectures and distributions. Note that the effort to fulfil most needs increases astonishingly because of the variety of architectures and distributions, thus many projects don't publish pre-compiled packages, or limit them to a subset of the possible target systems.

Examples here include the Debian format (.deb), as used by its native package manager (dpkg), and the RedHat format (.rpm), which requires a tool with the same name. Both are built upon tried-and-true archivers: the tar tool for .debs, and the cpio tool for .rpms.

With programs like apt-get, apt4rpm or urpmi, there are several possibilities to handle dependencies and archives.

2.2.2 Managed binary trees

For most beginners, this is what they see first when they install a Linux system: An abstract collection of tasks, which is used to setup the list of packages to be installed.

In Debian, the grandfather of this is dselect, which intersects the available packages into categories like Administration or Documentation. An even higher granularity is achieved by splitting up the package lists into free, free-yet-usable-only-with-non-free (contrib), and non-free parts. Note that both schemas are included within the packages, which fit together so well because of the Debian policy.

An independent tool is tasksel, which ships with its own lists and associated packages. It is already flexible enough to contain sections such as Chinese Environment or TeX Environment, and more sections can surely be added easily.

RPM-based distributions also each have their abstraction layers. This includes the series used in SuSE's yast system, along with their predefined choices such as Minimal System or Developer System.

Yet another way is to use distribution-independent tools such as Ximian's Red Carpet, or Trove-based selection systems as used by SourceForge.

This is where MatrixDB got most influence from.

Chapter 3

The universal approach

All packages have several properties, which determine if a user wants to install them or not. For example, when searching for an email client, the first step is to exclude all those packages which do not contain an email client. Due to the nature of Free Software, the remaining package list is not huge anymore, but still large. After all, the goal is to reduce this list to exactly one package, for hardly anyone uses more than one email client most of the time on the same computer.

Therefore, the second step is to figure out what other criterias can be used to reduce the number of packages. Here, it could be the environment: There are text-based email clients, graphical ones and web-based ones. Let's assume the user is happy with a graphical environment under X11 already, and thus selects the second choice.

This does still not lead to the goal. He is presented with quite a few programs, one is based on KDE and Qt libraries, the next one on GNOME and Gtk+ ones, and another one needs Tcl/Tk. In this case, it will mostly be the user's desktop environment which will lead to the final package, although a personal interest in one of them might of course dominate.

The universal approach is now to build up a chain of filters which are applied successively, to the initial list of packages first and then always to the previously resulting list (or result set, for that matter).

Chapter 4

Substitutions

To prove that the universal approach of MatrixDB is indeed usable universally, several already existing tools are to be simulated with it.

4.1 Substituting tasksel

Tasksel provides package lists so that choices do not collide. It can be viewed as a set of sets, which may or may not fill out the super set, being formed by all packages.