# Yum (Yellowdog Updater, Modified) HOWTO

Robert G. Brown, rgb at phy.duke.edu Jonathan Pickard,fatboy at techno.co.za 0.3, 2003-09-24

This is a HOWTO for Yum: the Yellowdog Updater, Modified. Yum is an automatic updater and package installer/remover for rpm-based systems. It automatically computes dependencies and figures out what things should occur in order to safely install, remove, and update rpm packages. Yum also efficiently and easily retrieves information on any package installed or available in a repository to the installer. Yum makes it easier to maintain groups of machines without having to manually update each one using rpm or other tools. Yum can manage package groups, multiple repositories, fallback repositories and more to permit centralized package management performed by just one or two individuals to scale over an entire organization. **Note Well!** This HOWTO is in a state of total flux during its initial pre-release development. Lots of sections are empty, others are wrong, all is in a state of mediocre organization. Nevertheless, I cherish feedback from anybody on the basis of any snapshot you happen to see.

## Contents

**1 Introduction** **3**

1.1 What Yum Can Do . . . . . 4

1.2 How Yum Works . . . . . 5

1.3 Yum, RPM, and Red Hat . . . . . 8

1.4 A Plea for Karmic Balance . . . . . 9

1.5 RPM Repository Madness . . . . . 9

1.6 Copyright . . . . . 11

1.7 Disclaimer . . . . . 11

1.8 News . . . . . 12

1.9 Credits/Acknowledgements . . . . . 12

**2 HOWTO Organization** **12**

2.1 How To Avoid Reading All of this HOWTO . . . . . 12

**3 Useful Links:** **13**

**4 Preliminaries** **13**

4.1 Superuser Requirements . . . . . 13

4.2 Getting Yum . . . . . 14

4.2.1 Downloading the Yum RPM . . . . . 15

4.2.2 Downloading the Yum Tarball . . . . . 15

4.2.3 Anonymous CVS . . . . . 15

**5 Planning a Yum Repository** **16**

5.1 A Single Repository (Using Rsync) . . . . . 18

5.1.1 Base Repository . . . . . 19

5.1.2 Updates Repository . . . . . 21

5.2 More Repositories and Servers . . . . . 21

**6 Setting up an FTP server for yum.** **22**

6.1 Download the RPM . . . . . 22

6.2 Install the ftp server. . . . . . 22

6.3 Edit the vsftpd.conf file . . . . . 23

6.4 Start the server . . . . . 23

6.5 Testing the FTP server . . . . . 24

**7 Building Yum** **24**

7.1 Creating a Yum RPM as Root . . . . . 25

7.2 Creating a Yum RPM as a User . . . . . 25

**8 Installing Yum** **25**

**9 Yummifying your servers: yum-arch** **26**

9.1 Non technical explanation. . . . . . 26

9.2 Creating headers for an ftp server. . . . . . 26

9.3 Advanced Commands . . . . . 28

9.3.1 check dependencies and conflicts in tree (-d) . . . . . 28

9.3.2 Potential solutions for conflicts. . . . . . 29

9.3.3 Potential solutions for dependancy issues. . . . . . 29

9.3.4 make output verbose. (-v and -vv) . . . . . 29

9.3.5 do not generate headers. (-n) . . . . . 30

9.3.6 make output more quiet. (-q) . . . . . 30

9.3.7 check pkgs with gpg and md5 checksums (-c) cannot be used with -n (obviously) . . . 30

9.3.8 compress headers using gzip. (-z) . . . . . 31

9.3.9 pay attention to symlinks (Default is to ignore symlinks). (-l) . . . . . 31

9.3.10 ...all you really need to do . . . . . 31

**10 Setting up the Yum Client** **31**

10.1 /etc/yum.conf . . . . . 31

10.2 /etc/cron.daily/yum.cron . . . 34
10.3 /var/cache/yum . . . 34
10.4 Distributing the Configuration . . . 34
10.4.1 Rebuilding the yum RPM with them ready to rock . . . 34
10.4.2 Push tools . . . 34

**11 Using the Yum Client 34**

11.1 Privileged Yum Commands The following commands alter the filesystem by installing, updating, deleting files. The first time yum is run, they will also download and cache all the header files in all the repositories in /etc/yum.conf, wich also requires root privileges. Thereafter the unprivileged commands will generally work for all users, but the commands in the subsections below will only work as root. . . . 35
11.1.1 Installing Packages with Yum . . . 35
11.1.2 Updating Packages with Yum . . . 35
11.1.3 Deleting Packages with Yum . . . 36
11.1.4 yum upgrade . . . 37
11.1.5 Yum Group Install/Update Commands . . . 38
11.2 Informational Yum Commands . . . 39
11.2.1 Listing Available and Installed Packages with Yum . . . 39
11.2.2 Getting information from packages with yum . . . 39
11.2.3 Listing the Contents of a Package with Yum . . . 39
11.2.4 Searching for a file in packages installed or uninstalled . . . 39

**12 Building RPMs for yum repositories 39**

**13 Security 39**

**14 Etc. 39**

**15 Further Information 39**

# 1 Introduction

*Yum* `http://www.linux.duke.edu/projects/yum` is a tool for automating package maintenance for a network of workstations running any operating system that use the *Red Hat Package Management (RPM) system* `http://www.rpm.org/` for distributing packaged tools and applications. It is derived from yup, an automated package updater originally developed for *Yellowdog Linux* `http://www.yellowdoglinux.com/` , hence its name: yum is "Yellowdog Updater, Modified".

Yup was originally written and maintained by Dan Burcaw, Bryan Stillwell, Stephen Edie, and Troy Bengegerdes of Yellowdog Linux (an RPM-based Linux distribution that runs on Apple Macintoshes of various generation). Yum was written and is currently being maintained by Seth Vidal and Michael Stenner, both of Duke University, although as an open source GPL project many others have contributed code, ideas, and bug fixes (not to mention documentation:-). The yum link above acknowledges the (mostly) complete list of contributers, as does the AUTHORS file in distribution tarball.

Yum is a Gnu Public License (GPL) tool; it is freely available and can be used, modified, or redistributed without any fee or royalty provided that the terms of its associated license are followed.

## 1.1 What Yum Can Do

Yum (currently) consists of two tools; yum-arch, which is used to construct an (ftp or http) repository on a suitable server, and yum, the general-purpose client. Once a yum repository is prepared (a simple process detailed below) any client permitted to access the repository can install, update, or remove one or more rpm-based packages from the repository.

Yum's "intelligence" in performing updates goes far beyond that of most related tools; yum has been used successfully on numerous occasions to perform a "running upgrade" of e.g. a Red Hat 7.1 system directly to 7.3 (where the probability of success naturally depends on how "customized" the target system is and how much critical configuration file formats have "drifted" between the initial and final revisions - YMMV).

In addition, the yum client encapsulates various informational tools, and can list rpm's both installed and available for installation, extract and publish information from the rpm headers based on keywords or globs, find packages that provide particular files. Yum is therefore of great use to *users* of a workstation, either private or on a LAN; with yum they can look over the list of available packages to see if there is anything "interesting", search for packages that contain a particular tool or apply to a particular task, and more.

Yum is designed to be a client-pull tool, permitting package management to be "centralized" to the extent required to ensure security and interoperability even across a broad, decentralized administrative domain. No root privileges are required on yum clients – yum requires at most anonymous access (restricted or unrestricted) *from the clients* to a repository server (often one that is maintained by a central – and competent – authority). This makes yum an especially attractive tool for providing "centralized" scalable administration of linux systems in a decentralized network management environment, where a mix of machines maintained by their owners and by a variety of network managers naturally occurs (such as a University).

One of yum's most common uses in any LAN environment is to be run from a nightly cron script on each yum-maintained system to update *every rpm package on the system* safely to the latest versions available on the repository, including all security or operationally patched updates. If yum is itself installed from a rpm custom-preconfigured to perform this nightly update, an entire campus that installs its systems from a common repository base can achieve near complete consistency with respect to distribution, revision, and security. Security and other updates will typically appear on all net-connected clients no more than 24 hours after the an updated rpm is placed on the repository by its (trusted) administrator who requires no root-level privileges on any of the clients.

Consequently with yum a single trusted administrator can maintain a trusted rpm repository (set) for an entire University campus, an entire corporation, an entire government laboratory or institution. Alternatively, responsibility for different parts of a distribution can be split up safely between several trusted administrators on distinct repositories, or a local administrator can add a local trusted repository to overlay or augment the offerings of the campus level repositories. All systems at a common revision level will be consistent and interoperable to the extent that their installed packages (plus any overlays by local administrators) allow. Yum is hence an amazingly powerful tool for creating a customized repository-based package delivery and maintenance system that can scale the work of a single individual to cover thousands of machines.

And it's free. It just doesn't get any better than that....

## 1.2 How Yum Works

To understand how yum works it helps to define a few terms:

- **server:** The term server generally refers to the physical system that provides access one or more ways to a repository. However when yum was first developed there was generally only one server per repository and server was often used as more or less of a synonym for repository. We will use it below *only* in the former sense – as a reference to a particular web, ftp, nfs server that provides access to a repository, not as the repository itself.
- **repository:** A repository is a collection of rpms under some sort of filesystem tree. For most purposes associated with yum, the repository will have two more important characteristics. It has had the command *yum-arch* run on the tree, creating a "headers" directory containing header and path information to all the rpm's under the tree, and is accessible by URL (which means as one or more of http://my.web.server.ext/path, ftp://my.ftp.server.ext/path, file://full/file/path to the repository tree).
- **serverid:** As noted above, there used to be a more or less one to one correspondance between servers and repositories in early versions of yum. However, this correspondance is not *many to many*. A single repository can be mirrored on many servers, and a single server can hold many repositories. When organizing "robust" access to repositories (which means providing URL's to the same repository on fallback servers in case the primary server of a repository is down) it is now necessary to label the repository with some sort of unique id that obviously cannot be the server or repository alone. The serverid is thus the unique label used in yum.conf to indicate that all the repositories given under a single baseurl are (presumably) mirrors of one another.
- **RPM:** This stands for "Red Hat Package Manager", the toolset developed by Red Hat for distributing and maintaining "packages" of tools, libraries, binaries, and data for their linux distribution. It is fully open source and is currently the basis for many linux distributions other than Red Hat. When the documentation below speaks of "an rpm" it refers to a single package, usually named packagename-version.rpm. To understand how yum functions, it is necessary to understand a bit about the structuring of rpm's.

An rpm consists of basically three parts: a header, a signature, and the (generally compressed) archive itself. The header contains a complete file list, a description of the package, a list of the features and libraries it provides, a list of tools it requires (from other packages) in order to function, what (known) other packages it conflicts with, and more. The basic rpm tool needs information in the header to permit a package to be installed (or uninstalled!) in such a way that:

- Installing the package breaks none of the already installed packages (recursively, as they may need packages of their own to be installed).
- All the packages that the package requires for correct operation are also (or already) installed along with the selected package, recursively.

- A later version of the package does not (accidentally) replace an earlier version of the package.

Note that a similar list applies to uninstallation; removing a package must not break any packages left behind, for example.

This process is generically known as "resolving package dependencies" and is one of the most difficult parts of package management. It is quite possible to want to install a packaged tool that requires two or three libraries and a tool. The libraries in turn may require *other* libraries, the tool other tools. By the time you're done, installing the package may require that you install six or eight *other* packages, *none* of which are permitted to conflict or break any of the packages that are already there or will remain behind.

If you have ever attempted to manage rpm's by hand, you know that tracking down all of the headers and dependencies and resolving all conflicts is *not easy* and that it actually becomes *more* difficult in time as a system manager updates this on one system, that on another, rebuilds a package here, installs something locally into /usr/local there. Eventually (sometimes out of sheer frustration) an rpm is –force installed, and thereafter the rpm database itself on the system itself is basically inconsistent and *any* rpm install is likely to fail and require –force-ing in turn. Entropy creeps into the network, and with it security risks and dysfunction.

Yet *not* updating packages is also a losing situation. If you leave a distribution based install untouched it remains clean. However, parts of it were likely *broken* at the time of install – there are always bugs even in the most careful of major distributions. Some of those bugs are security bugs, and as crackers discover them and exploits are developed it rapidly becomes a case of "patch your system or lay out the welcome mat for vermin". This is a global problem with *all* operating systems; even Windows-based systems (notorious for their vulnerability to viruses and crackers) can be made reasonably secure if they are rigorously kept up to date. Finally, users come along and demand THIS package or THAT package which are crucial to their work – but not in the original, clean, consistent installation.

In balance, any professional LAN manager (or even humble standalone linux workstation owner) has little choice; they *must* have some sort of mechanism for updating the packages already installed on their system(s) to the latest, patched, secure, debugged versions and for adding more packages, including ones that may not have been in the distribution they relied upon for their original base install. The only questions are: what mechanism should they use and what will it cost them (in time, hassle, learning curve, and reliability as well as in money). Let us consider the problem:

In a typical repository, there are a *lot* of packages (order of 1000), with a *lot* of headers. About 700 packages are actually installed on the system I'm currently working on. However, the archive component of each package, which contains the actual binaries and libraries and documentation installed, is much larger – the complete rpm is thus generally two to four *orders of magnitude* larger than the header. For example, the header for Open Office (a fairly large package) total about 100 *kilobytes* in size. The rpm itself, on the other hand, is about 30 *megabytes* in size. The header can be reliably delivered in a tiny fraction of a second over most networks; the rpm itself requires *seconds* to be delivered over 100BT, and minutes to be delivered over e.g. DSL, cable, or any relatively slow network. One occupies the physical server of a repository for a tiny interval; the other creates a meaningful, sustained load on the server. All of these are important considerations when designing or selecting an update mechanism intended to scale to perhaps thousands of clients and several distinct repositories per physical server.

Early automated update tools either required a locally mounted repository directory in order to be able to access all of the headers quickly (local disk access even from a relatively slow CD-ROM drive, being fast enough to deliver the rpm's in a timely way so that their headers could be extracted and parsed) or required that each linked rpm be sent in its entirety over a network to an updating client from the repository just so it could read the header. One was locally fast but required a large commitment of local disk resources (in addition to creating a new problem, that of keeping all the local copies of a master repository synchronized). The other was very slow. Both were also network resource intensive.

This is the fundamental problem that yum solves for you. Yum splits off the headers on the *repository side* (which is the job of its only repository-side tool, yum-arch). The headers themselves are thus available to be downloaded separately, and quickly, to the yum client, where they are typically cached semi-permanently in a small footprint in /var/cache/yum/serverid (recall that serverid is a label for a single repository that might be mirrored on several servers and available on a fallback basis from several URL's). Yum clients also cache (space permitting or according to the requirements and invocation schema selected by the system's administrator) rpm's when they are downloaded for an actual install or update, giving a yum client the best of *both* the options above – a local disk image of (just the relevant part of) the repository that is automatically and transparently managed *and* rapid access to just the headers.

An actual download of all the headers associated with packages found on your system occurs the first time a yum client is invoked and thereafter it adds to or updates the cached headers (and downloads and caches the required rpm's) only if the repository has more recent versions or if the user has deliberately invoke yum's "clean" command to empty all its caches. All of yum's dependency resolution then proceeds from these cached header files, and if for any reason the install or update requires an rpm already in the cache to be reinstalled, it is immediately available.

As a parenthetical note, the author has used yum's caches in a trick to create a "virtual" update repository on his homogeneous, DSL-connected home LAN. By NFS exporting and mounting (rw,no_root_squash) /var/cache/yum to all the LAN clients, once normal updates have caused a header or rpm to be retrieved for *any* local host, they are available to *all* the local hosts over a (much faster than DSL) 100BT NFS mount. This saves tremendously on bandwidth and (campus) server load, using instead the undersubscribed server capacity of a tiny but powerful LAN. Best of all, there "is no setup"; what I just described is the works. A single export and a mount on all the clients and yum itself transparently does all of the work.

However, it is probably better in many cases to use rsync or other tools to provide a faithful mirror of the repository in question and use yum's fallback capability to accomplish the same thing (single use of a limited DSL channel) by design. This gives one a much better capability of standing alone should update access go away on the "server" of the yum cache NFS exported across a LAN.

With the header information (only) handy on high-speed local media, the standard tools used to maintain rpm's are invoked by yum and can *quickly* proceed to resolve all dependencies, determine if it is safe to proceed, what additional packages need to be installed, and so forth. Note well that yum is designed (by a highly experienced systems administrator, Seth Vidal, with the help of all the other highly experienced systems administrators on the yum list) to *be safe*. It will generally *not* proceed if it encounters a dependency loop, a package conflict, or a revision number conflict.

If yum finds that everything is good and the package can be safely installed, removed, or updated, it can either be invoked in such a way that it does so automatically with no further prompts so it can run automagically from cron, or (the general default when invoked from a command line) it can issue a user a single prompt indicating what it is about to do and requesting permission to proceed. If it finds that the requested action is in fact *not* safe, it will exit with as informative an error message as it can generate, permitting the system's administrator to attempt to resolve the situation by hand before proceeding (which may, for example, involve removing certain conflicting packages from the client system or fixing the repository itself).

From the overview given above, it should be apparent that yum is potentially a powerful tool indeed, using a single clever idea (the splitting off of the rpm headers) to achieve a singular degree of efficiency. One can immediately imagine all sorts of ways to exploit the information now so readily available to a client and wrap them all up in a single interface to eliminate the incredibly arcane and complex commands otherwise required to learn *anything* about the installed package base on a system and what is still available. The yum developers have been doing just that on the yum list - dreaming up features and literally overnight implementing the most attractive ones in new code. At this point yum is very nearly the last thing you'll ever need to manage packages on any rpm based system once it has gotten past its original, distribution vendor based, install.

Indeed, it is now *so* powerful that it risks losing some of its appealing simplicity. This HOWTO is intended to document yum's capabilities so even a novice can learn to use it client-side effectively in a very short time, and so that LAN administrators can have guidance in the necessarily more complex tasks associated with building and maintaining the repositories from which the yum clients retrieve headers and rpm's.

Yum's development is far from over. Volunteers are working on a GUI (to encapsulate many of yum's features for tty-averse users). Some of yum's functionality may be split off so that instead of a single client command there are two, or perhaps three (each with a simpler set of subcommand options and a clear differentiation of functionality). The idea of making yum's configuration file XML (to facilitate GUI maintenance and extensibility) is being kicked around. And of course, new features are constantly being requested and discussed and implemented or rejected. Individuals with dreams of their own (and some mad python or other programming skills:-) are invited to join the yum list and participate in the grand process of open source development.

## 1.3 Yum, RPM, and Red Hat

Because yum invokes the same tools and python bindings used by e.g. Red Hat to actually resolve dependencies and perform installations (functioning as basically a supersmart shell for rpm and anaconda that can run directly from the local header cache) it has proven remarkably robust over several changes to the rpm toolset that have occurred since its inception, some of them fairly major. It is at least difficult for yum to "break" without Red Hat's own rpm installation toolset breaking as well, and after each recent major change yum has functioned again after a very brief period of tuneup.

It is important to emphasize, however, that yum is *not* a tool for administering Red Hat (only) repositories. Red Hat will be prominently mentioned in this HOWTO largely because we (Duke) currently use a Red Hat base for our campuswide linux distribution, maintain a primary (yum-enabled) Red Hat mirror, and are literally down the road a few miles from Red Hat itself. Still, if anything, yum is in (a friendly, open source) *competition* with Red Hat's own up2date mechanism and related mechanisms utilized by other distribution vendors.

So **Note Well:** Yum itself is designed for, and has been successfully used to support, **rpm repositories of any operating system or distribution that relies on rpm's for package management** and contains or can be augmented with the requisite rpm-python tools. Yum has been tested on or is in production on just about all the major rpm-based linuces, as well as at least one Solaris repository. Its direct conceptual predecessor (with which it shares many design features and ideas, although very little remaining actual code) is Yellowdog Linux's updater tool yup, which had nothing whatsoever to do with Red Hat per se. Yum truly is free like the air, and distribution-agnostic by deliberate design.

## 1.4 A Plea for Karmic Balance

It is worth taking a short moment here and put in a bit of a plug for the distribution providers, Red Hat, Mandrake, SuSE, Yellowdog and all the rest. Yum is a tool that (clearly) can completely short circuit some, if not all, of their expected/hopeful income streams. Using the methods described below, one can literally scale any distribution over the entire Internet, working directly from a mirror (of a mirror of a mirror...) from their original web distribution, in such a way that all maintenance is fully automated and yet makes the distribution provider absolutely no money for doing the toplevel maintenance on the base distribution itself.

There are obvious ethical and practical issues here. Ethically one should exchange value to remain in karmic balance with the world ecology, including the amorphous and chaotic one that encompasses all of linux. There are two ways to do so in the open source world: share in the work of providing the services or pay some money (to help pay for the time and profit of those that do the work for you).

Practically one should be aware that if a distribution provider doesn't make enough money to pay for the actual work and capital investment required to assemble, test, debug, maintain, and distribute the software (plus a fair profit) they will either go out of business or alter their business model in ways that make it more difficult for us all to work efficiently and scalably from their distribution without paying them some money.

For *both* reasons I would urge that even though one *can* obtain linux using the methodologies described in this HOWTO, install it on ten thousand systems in a single organization, and maintain it completely automagically with yum *for free*, one seriously consider meeting one's ethical and practical responsibilities and ensuring that you pay the global mind back, either way.

I personally do both. I'm writing this HOWTO, and have written GPL packages in the past and made them publically available. I have participated in all sorts of linux development processes and am on a dozen high level linux lists. I *still* make sure that I buy something inexpensive from Red Hat (my current primary distribution) every two or three years so that they make a buck or two for every one of the systems I personally run in my house, per year.

It would be lovely if the primary distribution vendors made this easier. Obviously, I install via dulug and maintain via yum, so the RH 9 box set I purchased was a complete waste (and I'll cheerfully give it away if I can find anybody to give it to). I'd have rather found a paypal link on the RH website where one could utterly voluntarily kick in a payment and NOT get a damn thing back from them – no updates, no free support services, nothing at all but continued *free* access to their distribution via the chain of mirrors I use to install and maintain it. Pure revenue for them, pure karmic peace for me.

## 1.5 RPM Repository Madness

A moment or two of meditation upon dependency resolution should suffice to convince one that Great Evil is possible in a large rpm repository. You have hundreds, perhaps thousands of rpm packages. Some are commercial, some are from some major distribution(s), others are local homebrew. What if, in all of these packages built at different times and by different people, you ever find that there exist rpm's such that (e.g.) rpm A requires rpm B, which conflicts with rpm C (already installed)? What if rpm A requires rpm B (revision 1.1.1) but rpm B (revision 1.2.1) is already installed and is required in that revision by rpm C (also already installed)? It is entirely possible to assemble an "rpm repository from hell" such that nearly any attempt to install a package will break something or require something that breaks something.

(As yet another parenthetical note, this was the thing that made many rpm-based distribution users look at

Debian with a certain degree of longing. Apt untangles all of this for you and works entirely transparently from a single distribution "guaranteed to be consistent", and provides some lovely tools (some of which are functionally cloned in yum) for package management and dependency resolution. However, as is made clear on the yum site, yum is a *better* solution in many ways than apt or, for that matter, Current or up2date. I believe that the designers are working fairly aggressively to make sure it stays that way.)

A cynical (but correct) person would note that this was why rpmfind and other rpm "supertools" ultimately failed. Yes, rpmfind could locate any rpm on the planet in its superrepository a matter of a few seconds, BUT (big but) resolving dependencies was just about impossible. If one was lucky, installing an e.g. Mandrake rpm on a Red Hat system that used SuSE libraries rpm's would work. Sometimes one required luck to install the Red Hat rpm's it would find on a Red Hat system, as they were old or built with non-updated libraries. Sometimes things would "kind of work". Other times installing an rpm would break things like all hell, more or less irreversibly.

Untangling and avoiding this mess is what earns the major (rpm-based or not) linux distribution providers their money. They provide an entire set of rpm's (or other packages) "all at once" that are guaranteed to be consistent in the distribution snapshot on the CD's or ISO images or primary website. All rpm's required by any rpm in the set are in the set. No rpm's in the provided set conflict with other rpm's in the set. Consequently any rpm in the set can be selected to be installed on any system built from the distribution with the confidence that, once all the rpm dependencies are resolved, the rpm (along with its missing dependencies) can be successfully installed. The set provided is at least approximately complete, so that one supposedly has little incentive or need to install packages not already in the distribution (except where so doing requires the customer to "buy" a more expensive distribution from the vendor:-).

In the *real* world this ideal of consistency and completeness is basically never achieved. All the distributions I've ever tried or know about have bugs, often aren't totally consistent, and certainly are not complete. A "good" distribution can serve as a base for a repository and support e.g. network installs as well as disk or CD local installs, but one *must* be able to add, delete, update packages new and old to the repository and distribute them to all the systems that rely on the repository for update management both automatically and on demand.

Alas, rpm itself is a *terrible* tool to use for this purpose, a fact that has driven managers of rpm-based systems to regularly tear their hair for years now. Using rpm directly to manage rpm installs, the most one can do is look one step ahead to try to resolve dependencies. Since dependency *loops* are not at all uncommon on real-world repositories where things are added and taken away (and far from unknown even in box-set linux distributions that are supposed to be dependency-loop free) one can literally chase rpm's around in loops or up a tree trying to figure out what has to be installed before finally succeeding in installing the one lonely application you selected originally.

rpm doesn't permit one to tell it to "install package X and anything else that it needs, after YOU figure out what that might be". Yum, of course, does.

Even yum, though, can't "fix" a dependency loop, or cope with all the arcane revision numbering schemes or dependency specifications that appear in all the rpm's one might find and rebuild or develop locally for inclusion in a central repository. When one is encountered, a Real Human has to apply a considerable amount of systems expertise to resolve the problem. This suggests that building rpm's from sources in such a way that they "play nice" in a distribution repository, while a critical component of said repository, is *not a trivial process*. So much so that many rpm developers simply do not succeed.

Also, yum achieves its greatest degree of scalability and efficiency if *only* rpm-based installation is permitted on *all* the systems using yum to keep up to date. Installing locally built software into /usr/local becomes

Evil and must be prohibited as impossible to keep up to date and maintained. Commercial packages have to have their cute (but often dumb) installation mechanisms circumvented and be repackaged into some sort of rpm for controlled distribution.

Consequently, repository maintainers must willy-nilly become rpm builders to at least some extent. If SuSE releases a lovely new tool in source rpm form that isn't in your current Red Hat based repository, of course you would like to rebuild it and add it. If your University has a site license for e.g. Mathematica and you would like to install it via the (properly secured and license controlling) repository you will need to turn it into an rpm. If nothing else, you'll need to repackage yum itself for client installations so that its configuration files point to *your* repositories and not the default repositories provided in the installation rpm's /etc/yum.conf.

For all of these reasons an entire section of this HOWTO is devoted to a guide for repository maintainers and rpm builders, including some practices which (if followed) would make dependency and revision numbering problems far less common and life consequently good.

In the next few sections we will see where to get yum, how to install it on the server side, and then how to set up and test a yum client. Following that there will be a few sections on advanced topics and design issues; how to set up a repository in a complex environment, how to build rpm's that are relatively unlikely to create dependency and revision problems in a joint repository, how to package third party (e.g. site licensed) software so it can be distributed, updated, and maintained via yum (linux software distributors take note!) and more.

## 1.6 Copyright



## 1.7 Disclaimer

Use the information in this document at your own risk. I disavow any potential liability for the contents of this document. Use of the concepts, examples, and/or other content of this document is entirely at your own risk.

Naming of particular products or brands should not be seen as endorsements.

You are strongly recommended to take a backup of your system before major installation and backups at regular intervals.

### 1.8 News

This is the first release of this document, so there isn't much news.

Eventually we will hope that the very latest version number of this document can be obtained from a URL like *http://www.linux.duke.edu/projects/yum/yum_HOWTO.html* `http://www.linux.duke.edu/projects/yum/yum_HOWTO.html`

(which probably doesn't work yet).

### 1.9 Credits/Acknowledgements

```
Greg Wildman          gregw at techno.co.za
Stein Gjoen           sgjoen at nyx.net
Russ Herrold
Seth Vidal
Michael Stenner
```

(and others on the yum list).

In particular rgb gratefully acknowledges the help of Russ Herrold for writing the original, non-HOWTO online yum documentation from which this is loosely derived, as well Seth Vidal and Michael Stenner, who wrote and continue to maintain yum, including the base documentation from which this HOWTO is derived.

Any comments or suggestions can be mailed to the : *yum mailing list* . You might visit the *yum mailing list* `https://lists.dulug.duke.edu/mailman/listinfo/yum` website.

## 2 HOWTO Organization

### 2.1 How To Avoid Reading All of this HOWTO

There are three general categories of reader for whom this HOWTO is designed:

- **Standalone System Users:** If you own one or more systems, have no interest in setting up a repository of your own and just want to learn to use yum to shop for and install new software packages and keep your systems up to date, this is you. You may skip around to the sections on preliminaries and installing yum, yum.conf and the yum client itself. As you have time or interest and more than one system you might consider skimming the other sections as well – it is easier to set up a repository mirror than you might think, and besides, you can never know too much about a core toolset. You may or may not wish to join the yum mailing list for additional support getting started – once you've gotten started, you probably won't need it.

- **Systems Administrators of Workstation Networks** If you own or are responsible for one or more systems and wish to set up one or more repositories, you will need to read or at least skim the entire document. This HOWTO is in fact designed primarily with you as an audience in mind. Parts of it (inserted to support less knowledgeable readers) can be skipped, but you'll know them when you see them. You may or may not wish to or need to join the yum development mailing list for additional support or to contribute to its development.

- **Developers** If you are a software developer who is interested in using yum as a direct distribution agent or in contributing to the development of yum itself, you too will need to read or skim the entire document, but for you the document is likely to be inadequate. You should almost certainly join the yum list linked to the primary yum website linked below.

# 3 Useful Links:

The following are links that may be useful to individuals learning yum:

- **Primary Yum Website:**

  *http://www.linux.duke.edu/projects/yum* `http://www.linux.duke.edu/projects/yum`

- **Yum HOWTO (latest version):**

  *http://www.phy.duke.edu/~rgb/General/yum_HOWTO.php* `http://www.phy.duke.edu/~rgb/General/yum_HOWTO.php`

- **Yum Article:**

  *http://www.phy.duke.edu/~rgb/General/yum_articicle.php* `http://www.phy.duke.edu/~rgb/General/yum_article.php` This is a short article on yum, a sort of "getting started".

- **Duke University Linux Users Group (DULUG) Red Hat repository mirrors:**

  *http://mirror.dulug.duke.edu/pub/yum-repository/redhat* `http://mirror.dulug.duke.edu/pub/yum-repository/redhat` This repository collection is yummified and support Red Hat Linux 7.1 to 9. Red Hat Linux is deprecated in favor of Fedora for most users as Red Hat will no longer support these versions as of next year (2004). This is currently the repository the default yum.conf points to, so that yum should work more or less automatically out of the RPM box for most Red Hat users.

# 4 Preliminaries

## 4.1 Superuser Requirements

Linux and other Unices prevent ordinary users from being able to install, remove, or modify software, libraries, or important configurational information. On a networked, professionally run installation of workstations the need for this sort of control to ensure security and functionality is obvious. On a privately owned and controlled workstation, the need is less obvious but just as profound. Even there the owner of the workstation needs to take extreme care whenever working with the fundamental system configuration and package installation or they will likely break the system or leave it vulnerable to crackers.

To ensure that an adequate degree of care is exercised when doing things that could render a system *unusable* or *vulnerable*, linux and Unices require the owner of such a workstation to assume *root privileges*, usually by means of the su command. In order to execute the privileged commands to add or remove software from your system, you must therefore become root. To do this, enter (in a tty window such as an xterm):

```
$ su
```

and enter the root password when prompted to do so. The prompt should become the sharp sign '#'

```
# yum list
```

to remind you to be sharp as root.

**Warning!** It is *essential* that you heed this advice to "be sharp" as you can **destroy your system's installation** with a careless keystroke, especially one involving wildcards such as '*' or '?'. Certain yum commands (notably upgrade and remove) can have unexpected side effects depending on the whims of the maintainers of RPMs in the repository, so caution and testing are advised. Be **certain** that you understand what a command will do, including side effects, before invoking it, especially with commands that delete files.

Some yum commands will actually work fine for ordinary, unprivileged users. In general these are the informational yum commands, discussed next. Note that the unprivileged commands *do* require that the yum header cache exist, so they will *not* work at all correctly if run before root has run a yum command to create current images of the header files.

## 4.2 Getting Yum

There are several ways of "getting yum":

- If you are *only* trying to obtain a copy of yum to install on your personal workstation so you can use it as a client, directing it for updates to one of the growing number of public repositories that support yum-based access, then you can download the prebuilt rpm's from the yum website linked below.
- If you are setting up a repository to serve a LAN (or even a WAN) *or* you are planning to install yum for use as a client only on a LAN, again directing it to one of the public repositories that support yum, and are not interested in participating actively in yum development, then you should get the yum tarball (.tgz compressed source directory) from the yum website linked below.
- Finally, if you are setting up yum for a LAN and either want to participate in yum's ongoing development (joining the yum list to get help and submit bug reports and patches) or encounter difficulty getting the stable version of yum downloaded from the yum website working (perhaps because your system is very new or not one of the distributions already well-supported), you will likely want to access the CVS repository and get a tarball of the current yum snapshot.

Each of these possibilities is detailed in the subsections below.

### 4.2.1 Downloading the Yum RPM

If your goal is just to get yum and install it on a client, you will likely just want to get yum in a ready-to-install-and-run rpm from from the main yum website. One can select and download the appropriate rpm for your architecture (*which matters*) from the yum website *http://www.linux.duke.edu/projects/yum* `http://www.linux.duke.edu/projects/yum` .

To explain the importance of matching architecture for a tool written in a presumably agnostic and slowly varying scripting language, note that yum is written in python (the agnostic and slowly varying language:-) and uses the python rpm bindings, the same as Red Hat's anaconda and up2date. Ahh, good news and bad news, same as always.

The good news is that the basic python code is indeed quite stable, and using the python-rpm bindings makes yum relatively unlikely to break, at least for long, as Red Hat has a vested interest in making these binding work for their own products.

The bad news is that as rpm changes (fairly rapidly in recent releases) the bindings change and yum needs revision. Hence there are thus several versions of yum available for download at this time, each matching a quite recent revision of rpm and (not coincidentally) a release number of Red Hat Linux.

Fortunately, there is a short table of download links on the server above where you can find ready to install rpm's for yum built for each of the recent major linux releases. If you have difficulty picking, ask on the yum list for help or try using the tarball of yum sources to build the right version automagically. Or both!

**Note Well:** The yum rpm's you will download from this site have an /etc/yum.conf that is preconfigured to point to a public repository that supports yum, probably *http://mirror.dulug.duke.edu* `http://mirror.dulug.duke.edu` . Naturally, if the entire Internet got yum and started updating clients from this site, at some point it would pretty much get hammered to where it no longer worked. DULUG is not google, and this is a single not-particularly powerful server, not a web farm. You may well want to look over the list of public yum repositories and edit /etc/yum.conf to use one of the alternatives, or (if you are a LAN manager with a publically accessible webserver) set up a repository of your own and add it to the list, following the instructions in this HOWTO!

### 4.2.2 Downloading the Yum Tarball

To set up a repository of your own to serve a private LAN or public WAN or both, you will need the yum tarball so that you can build your own yum rpm with its configuration files and %post installation instructions customized for your site (following the instructions given in the next couple of sections). If you are new to yum, you should likely start with the stable tarball, not the development tree, and join the yum list. You'll know when it is time to move up and start using the public CVS repository to get the latest tarball snapshot instead of using the stable tarball.

To get the stable tarball is simple. Just visit the yum website,

*http://www.linux.duke.edu/projects/yum* `http://www.linux.duke.edu/projects/yum` , and download it. Then skip ahead to the sections on planning a repository and building a custom yum rpm.

### 4.2.3 Anonymous CVS

As noted above, it is also possible to get yum directly from the CVS tree on the linux@duke repository. Most "normal humans" won't need or want to do this – you are accessing the raw development tree, which might have cool new features or support a broader range of cutting edge distributions and releases, but which also might have bugs. Even immediately fatal bugs – some snapshots you might grab will simply be pre-broken and need immediate replacement.

To use the CVS repository you don't quite need mad skills, but you *do* need to be familiar with and comfortable with the open source development process. Joining the yum mailing list is *essential* for example, and in fact if you are reading this in preparation for doing a CVS download I'm going to hope that you've been on the list already for a while, and are grabbing the CVS-based snapshot fully understanding that if it doesn't work you'll submit an immediate bug report and wait for (or help find) a solution, not scream because it "breaks" your production network.

It should almost *never* break your production network because anybody who participates in this sort of development cycle knows better than to immediately deploy such a snapshot without a fair bit of testing on clients and servers where if it turns out to be broken "it doesn't matter".

Don't let these dire-sounding warnings deter you, though. The entire operating system you are using (presuming that you are using Linux) was built by precisely this process, and many hands contributing a small bit of work, intelligently directed, can create incredibly powerful tools. Come on in, the water's fine! Especially if you have mad python skills and some clever ideas...;-)

To pull yum from cvs do the following:

```
cd ~
mkdir cvsroot
cd cvsroot
cvs -d :pserver:anonymous@devel.linux.duke.edu:/cvsroot/yum/cvs/\
  login
cvs -d :pserver:anonymous@devel.linux.duke.edu:/cvsroot/yum/cvs/\
  get yum
```

To update it once you've gotten it:

```
cd ~
cd cvsroot
cvs -d :pserver:anonymous@devel.linux.duke.edu:/cvsroot/yum/cvs/\
  upd yum
```

[*Thanks to Russ Herrold for providing the above instructions*]

As noted above, it is *strongly recommended* that one become pretty thoroughly familiar with yum from using a stable release version and at least listening in on the yum list (which is mailman-managed and hence can easily be digested or throttled altogether) before tackling a CVS tree build. Knowing python is probably a good idea as well.

Once you have yum sources from either the stable tarball or the CVS tree, it is time to plan out your repository. This is detailed in the next section

# 5 Planning a Yum Repository

Before beginning, let us define a couple of terms. When we refer to a "server" below, let me remind you that we are referring to a piece of hardware. **Beware!** In e.g. "man yum.conf" a [server] is really a label for a *repository*, not a web, ftp or file server that might be one of several that provide mirrored access to the same repository. This is made clear later in the man page, where it is referenced in practice as [serverid]. A single physical "server" might well offer several repositories, each identified by a unique URL path on the server and by a unique [serverid] in /etc/yum.conf on the clients. Each [serverid] may similarly have several fallback URL's for the same repository (generally on different servers). This may initially be confusing, but it makes sense and provides for extremely robust operation.

As discussed in the Introduction, yum works by means of accessing a "yummified" repository – a URL-accessible path where yum-arch has been run on the server to create a special directory with a known relative path displacement from the RPMS directory used to actually provide the downloads, into which all of the split-off headers are placed so they can be separately downloaded and cached on the client side.

Simple as this is (really!) there is a *bit* of work that should be expended planning out a yum repository, especially if you are starting from scratch and don't already have an rpm repository in place to which you are planning to "add yum support". It is probably wise to start simple and work up to a complex system of repositories and servers gradually and as need dictates (many sites will ONLY need a single monolithic server, maintained by a single wise administrator, with a single repository apiece for each distribution supported).

However, it never hurts to know a bit about what *can* be done in terms of repository design. To meet specific needs of the fairly wide variety of systems persons already using yum and participating in its active development, some fairly slick features have been added: the ability to operate from multiple servers and repositories in an overlaid/prioritized fashion with failover, for example.

In a small LAN this is total overkill. If one is using yum to support an entire University, however, where each department has its own "special, must have" packages or (a more common case) some rpm's that set up e.g. /etc with the appropriate configuration for just that department, the ability to specify multiple repositories on different servers that are checked in a particular order lets those departments manage *just* the rpm's they require that are department specific or otherwise customized.

This keeps the manager of the central/primary repository(s) sane, as that individual does *not* have to juggle lots of departmental configuration rpm's with names like physics-stuff-0.1.1.noarch.rpm, biology-stuff-0.1.1.noarch.rpm,... and keeps the central/primary server secure, as the departmental lan managers do *not* need any sort of trusted access to the primary server. This has nothing to do with whether or not these departmental managers are fundamentally "trusted" – it is simply *always* a good idea to heavily restrict and regulate root-level access to a server the cracking of which de facto cracks an entire network! Putting rabid mastiffs outside the door to the server room isn't out of the question.

The following is an approximate (probably incomplete, as features are constantly being discussed on the yum list and added) list of some of the options and True Facts to consider when planning, building, or modifying a yum-based rpm repository (set):

- Yum can currently work from either http (recommended) or ftp or nfs/file servers. All that is required is that the repository be accessible via a URL from the client(s) in question.
- Yum is configured in /etc/yum.conf to automagically access a list of repositories...
- Which are just URL's – server paths to directories that contain rpms...
- According to a policy that determines the order they are checked...
- With failover, so rpms can be reliably delivered...
- And optional GPG signature checking, so rpms can be securely delivered...

- And control over architecture specificity, so one can load (e.g.) Athlon or i686 where it matters but i386 or noarch where it does not...
- And control over distribution specificity, so one doesn't accidentally update in a way that breaks things silently.
- In the True Facts category, since yum is *designed* to facilitate distributed maintenance (automated updates) of an rpm-based distribution of *something* – operating systems, complete distribution, or even just a finite set of packages on some system installed and maintained with completely different tools – the repository should be designed and built so that *it* remains up to date!

*Some* people won't need any of these bells and whistles. Start simple, yum's defaults are probably what you want anyway. *Most* people won't need more than one or two servers, perhaps a base distribution server (with primary repository images) plus an overlay server which might also mirror the primary repositories, used in a specified order, with GPG checking, for example. *Some* people, generally toplevel and local systems managers co-maintaining a repository network for a large organization, will use *all* of these features and even clamour for more on the yum list.

In the subsections below a strategy for planning out a yum repository/server set will be indicated that lets one smoothly move from a single server of a single repository (possibly originally mirrored from and updated from one of the existing distribution repositories) through to a really complex set of repositories on multiple servers that use all of the features.

## 5.1 A Single Repository (Using Rsync)

To set up your first repository, it probably makes sense to clone a repository from an existing server. Yum works to support pretty much any kind of rpm-based repository for any of the major Linux distributions across several hardware architectures, as well as for other operating systems that either already support rpm package management or upon which the rpm toolkits and python can be built. Unfortunately, that makes giving detailed instructions for EVERY ONE of those possibilities impossibly complex. It is probably better to give an example and leave it up to you to figure out what you need to do to set up a "leaping frog linux" distribution repository or a repository for all of the rpm's required to do molecular dynamics off of a common set of tools and data on a small cluster.

Ordinarily as a very first step one would pick a server/repository to mirror. Be cautious. Some repositories are public and their servers have lots of capacity. Others may require permission to use. Some will support rsync, others will require wget or worse as retrieval agents. Usually the repository itself will tell you what its policy is with respect to mirroring (and of course won't let you in at all if it is truly restricted), but it never hurts to ask if there is any doubt.

Rsync is probably the tool of choice for mirroring and maintenance. It only downloads files if it needs to and is very smart and (one hopes) reasonably secure. So we'll use rsync in our examples below. However, not everybody sets it up on their servers. So what do you do if the repository you wish to mirror doesn't support rsync?

At least one possibility (there are probably others) is to use wget. wget is a common tool that can be used to get whole branches, recursively, from a website in order to create a locally rereferenced copy or a mirror. It is beyond the scope of this HOWTO to detail all the steps at this point; but the wget documentation is fairly clear. Basically, one simply substitutes the appropriate wget invocation, (recursive and to an adequate depth and probably with links rereferenced, with the possibly different source url) for rsync in the examples below

For specificity only, then, the following example describes one way to set up a repository that is a mirror of the rsync-enabled dulug (primary, tier1) Red Hat mirror, and we'll even more specifically restrict ourselves to version 9 i386 and pretend that yum is *not* already installed on the repository.

We start by trying:

```
rgb@lilith|T:121>rsync mirror.dulug.duke.edu::
mirror.dulug.duke.edu
 - This rsync server is currently available to any/all people.
 - This is subject to change with little or limited notice.
 - We may in the not-so-distant-future restrict rsync access
   to tier2 red hat mirrors.


Modules:

archive         Everything
redhat-ftp      Red Hat FTP Site
redhat-base     Red Hat FTP Site
redhat-beta     Red Hat Linux beta releases
redhat-rawhide  Rawhide FTP Site
redhat-updates  Updates FTP Site
redhat-contrib  Red Hat, Inc. -- Contrib FTP Site



archive         Main Tree
redhat-ftp      Red Hat FTP Site
redhat-base     Red Hat FTP Site
redhat-beta     Red Hat Linux beta releases
redhat-rawhide  Rawhide FTP Site
redhat-updates  Updates FTP Site
redhat-contrib  Red Hat, Inc. -- Contrib FTP Site
```

Oh, good. Anonymous rsync is supported on this repository, and the repository's policy says we can go ahead, at least for now, and mirror any repository to be found on this site ourselves. However, it also notes that *quite possibly* access to this mirror in the near future will be restricted to tier2 mirrors (basically sites that let other people mirror them and hence REDUCE the load on the tier1 sites) if load gets to be more than it can handle, so perhaps you (who read this) might consider using a mirror of one of the tier2 mirrors instead, unless you are hoping to set up a site that will *be* a tier2 mirror and publically available in turn.

### 5.1.1 Base Repository

After a little bit of monkeying around with rsync and/or a web browser, we find the path we're looking for to a suitable "base" Red Hat 9 tree that will constitute our repository. In actual fact we'd probably want to mirror one level up from here even for a "version 9 only" repository, as one level up we'd find e.g. documentation and iso images for the release which we'd almost certainly like to have handy even if yum itself points to a repository path one directory down. For the purposes of this example, however, we go to our (already set up) web or ftp server and create a suitably named path to our new repository.

This path should probably contain something that indicates distribution and revision it contains as part of its eventual URL; http://whatever.org/base is probably a bad choice, but http://whatever.org/RedHat9/base might be ok, or something much longer if you have lots of distributions or revisions or other repositories on the server.

After creating the path, enter the directory and enter (for example):

```
rgb@lucifer|T:27#rsync -avH
mirror.dulug.duke.edu::redhat-ftp/redhat/linux/9/en/os .
mirror.dulug.duke.edu
 - This rsync server is currently available to any/all people.
 - This is subject to change with little or limited notice.
 - We may in the not-so-distant-future restrict rsync access
   to tier2 red hat mirrors.


Modules:

archive         Everything
redhat-ftp      Red Hat FTP Site
redhat-base     Red Hat FTP Site
redhat-beta     Red Hat Linux beta releases
redhat-rawhide  Rawhide FTP Site
redhat-updates  Updates FTP Site
redhat-contrib  Red Hat, Inc. -- Contrib FTP Site



receiving file list ... done
os/
os/i386/
os/i386/RedHat/
os/i386/RELEASE-NOTES-it.html
os/i386/dosutils/
os/i386/dosutils/fips15c/
os/i386/dosutils/fips20/
os/i386/images/
os/i386/RELEASE-NOTES-ja.html
...
```

(for a few hundred directories packages and files more) and we're off to the races.

Depending on your bandwidth, you will have a brand, spanking new Red Hat 9 repository in minutes to hours (however long it takes to transfer a few GB of RPM's and support materials). Obviously, a nearly identical process would work for any other distribution that has rsync-enabled originals or mirror servers. Don't forget to consider buying SOMETHING from the original distribution provider (if they sell anything) to help them make some money from providing the tremendous value you are about to distribute at zero marginal cost across a whole network of systems!

### 5.1.2 Updates Repository

The next thing to worry about is keeping the new base repository itself up to date. There are two levels to even this – one is rsync'ing the repository itself with the master on a periodic basis. This is accomplished with a simple cron script that changes to the appropriate directory and reruns the rsync command. rsync is smart enough to only download and replace files if they've changed. rsync can be run to either delete files that disappear from the master (remaining absolutely faithful to the master image) or to get new files and update changed ones but leave the old copies of ones that disappear. You have to decide which policy is right for your site, as both have risks.

You may want to quit there, and if the primary repository image already contains an "updates" path where updates automatically appear as bugs are fixed and security holes plugged, you may be able to. The repository just installed should support basic network installs (if the distribution does in the first place) and automated yum updates to the images in the repository once your clients are appropriately configured.

However, in many cases you *will* need to worry about keeping this base system correctly updated outside of the automatic updates provided by the distribution maintainer or vendor.

There are a couple of reasons for this. One is that not every distribution will handle updates the same way. Some distributions may be very aggressive and maintain updates that match and replace their distribution tree with a numbering/dating scheme that yum can understand and that changes rapidly as security, performance, and bug patches are released (Red Hat does). Others might replace files in their master distribution silently, after you've come to rely on a feature that suddenly goes away. Others might be lazy and not update their master distribution at all, or may have specialized tools for detecting and retrieving updates. Finally, even if your master distribution is very reliable and aggressively maintained, you may wish to *override* its offerings and update from a local copy of some of the rpm's, perhaps because they contain custom patches or local data.

One solution to these various dilemnas is to add an updates repository separate (different URL/path, recall, quite possibly on the same physical server). Here you can again choose to rsync to an updates server somewhere else, or you can create one all your own and split updates out of the distribution itself any way you like.

## 5.2 More Repositories and Servers

In addition to a separate update repository, you may well want (eventually) to add *still* more repositories, and possibly more servers (viewing a repository, recall as a specific path to a single URL source of rpm's not necessarily on a distinct server). Perhaps you want to make a games rpm repository available to everybody on campus, but not on the main server which is owned by philistines that view games as a waste of time. Perhaps you want to overlay department rpm's on top of a base distribution repository (set, including updates) on a central server.

The outline above, and the example of the more or less required base repository, should be enough to enable you to set up arbitrary URL pathways to new repositories for whatever purpose you like. However, yum cannot use these repositories *yet*. First one has to install yum on the server(s) and run the *yum-arch* command on each repository. This causes the rpm headers to be extracted and paths to be dereferenced in a way that makes yum ultimately very fast and efficient.

Once this step is completed, yum then allows clients, via options in yum.conf on the various clients, to access the repositories and servers in whatever order they like (or more likely, in the order specified in the yum client rpm they automatically loaded at install time from one of those repositories).

# 6 Setting up an FTP server for yum.

Setting up an FTP server is very simple. The steps are as follows:

- Download the rpm.
- Install the rpm.
- Edit the .conf file.
- Start the ftp server.
- Test the connection.

## 6.1 Download the RPM

There are many ftp servers available for Linux. Most ftp servers do the same thing so the choice is really up to you. The ftp server that I like to use is *vsftpd*. The rpm is usually available from the installation discs or can be downloaded from *rpmfind* `http://www.rpmfind.net` or just use *google* `http://www.google.com` . vsftpd is used by many large companies as the ftp server of choice and is very secure (it's part of the name so it must be true, right?).

## 6.2 Install the ftp server.

First check to see if VSFTPD is already installed on your machine, this is easily done by using:

```
root@cartman> rpm -q vsftpd
```

The system will tell you if the server is installed or not. If you get this message **'package vsftpd is not installed'** then you will need to install the ftp server.

First download the latest version of VSFTPD from your preferred mirror and save it to e.g. /tmp on the server. The ftp directory structure required for your repository is unlikely to exist yet so you will need to create the repository directories that you planned out above, for example:

```
root@cartman> mkdir -p /var/ftp/pub/9/updates/
```

(the -p flag tells mkdir to create the whole tree of directories as required).

To install/upgrade the ftp server run the following as root:

```
root@cartman>rpm -Uvh /tmp/vsftpd-1.1.3-8.i386.rpm
```

Note that one will want this rpm to be in a repository the server itself uses to yum update from in the long run. It is very likely to be in a primary distribution repository you mirror, but you may have to put it in a local/update repository you maintain yourself from some other source.

(You can of course use rpm -ivh vsftpd-1.1.3-8.i386.rpm to install the package if the package is not already installed. The flag -U is for upgrade and -i is for install. No big deal, they will both work if the package does not exist on your system, IMHO -U is just better practise. It is not a good idea to use rpm -i if a previous version of the package already exists on your system.)

## 6.3 Edit the vsftpd.conf file

After the ftp package has been installed you will need to edit the vsftp.conf file. This is usually found at **/etc/vsftpd/vsftpd.conf**. If it is not here then just run:

```
jdip@cartman>rpm -ql vsftpd
```

and look in the list where the .conf file is. To edit the .conf file you can use kate, gedit, vi or any other text editor. This is the configuration file for the ftp server. You will need to be root to change the file:

```
root@cartman>vi /etc/vftp/vsftp.conf
```

If your network is secure and behind a firewall then you can leave the following option in the .conf file. This option allows for anonymous ftp access to your server:

```
# Allow anonymous FTP? (Beware - allowed by default if you comment this out).
anonymous_enable=YES
```

You can also change the welcome message of the ftp server.

```
# You may fully customise the login banner string:
ftpd_banner=Welcome to yum FTP service.
```

If you want increased security for your ftp server then set the flag **anonymous_enable=NO**. This will force the user to log into the ftp server to get access to the packages. If you want to use this option then you will need to create a yum user on the server that can be used by the yum client to access the server. It is prudent to make users log into the ftp server, but if this is your private server then it may not be necessary.

Save the .conf file.

You will need to (re)start the service to activate the changes to the ftp server (see next section).

## 6.4 Start the server

If you installed VSFTPD from the rpm then VSFTPD can be started as a service:

```
root@cartman>service vsftpd restart
```

You should get this message:

```
Shutting down vsftpd:                              [  OK  ] or [ FAILED ]
Starting vsftpd for vsftpd:                        [  OK  ]
```

You will want your ftp server to start every time you start Linux so it is also prudent to run:

```
root@cartman>chkconfig vsftpd on
root@cartman>chkconfig --list vsftpd
```

You should get a message that looks like this:

```
vsftpd          0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

Your ftp server will now start every time you start Linux on this machine. The ftp server is up and waiting for connections.

### 6.5 Testing the FTP server

It is a good idea to test that the ftp server is working correctly. This is easily done by logging onto the ftp server:

```
jdip@cartman>ftp 127.0.0.1
Connected to 127.0.0.1 (127.0.0.1).
220 Welcome to yum FTP service.
Name (127.0.0.1:root): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

If you type **ls** at the prompt you will see that you are in the 'pub' directory. This is /var/ftp/pub. This path (and the full path of the repository you're setting up) are important to remember for when you use rsync to build the mirrors, for when you use yum-arch to "yummify" the repository (see below), and for setting up a local yum.conf for your local clients so that they can update from this ftp-based repository.

That is it. The ftp server is running and waiting for connections. Skip ahead to where it describes how to get and install yum and yummify the repository.

## 7 Building Yum

Before proceeding further, we need to have yum itself handy, specifically the yum-arch command and its current documentation. If you are working from the rpm's, you've probably already installed them on your repository (I mean actually installed the program, not necessarily inserted the rpm's into a server on the repository) and one or two test clients. If not, please do, and skip ahead to the sections on installing yum or setting up a server with yum-arch and creating a suitable /etc/yum.conf.

However, if you get the sources via tarball or from the CVS repository, you will have to locally build yum. If you plan to repackage it (basically required if you are setting up a repository ) so that yum clients automatically use the yum-based repositories you set up in their /etc/yum.conf, you will need the tarball (yum-*.tgz) anyway.

The steps required to transform the provided tarball into an rpm are given below. Note that many of these steps are *not yet fully documented* in the source README or INSTALL files and are a major reason a HOWTO is sorely needed by the project. Most of yum's current systems manager users are already sufficiently expert to be able to build rpm's without additional instructions, but of course many who would *like* to use yum are not, and in any event it never hurts to document a moderately complicated process even for the experts.

Experts can also disagree. The steps below are ONE way of proceeding, but there are many others. Some managers will be working in monolithic (topdown) management models where they have root control of all clients and will prefer to push /etc/yum.conf out to the clients directly, not de facto pull it onto clients during an install from a repository where it is available, preconfigured for the site in rpm form. Tools exist to make this simple enough (cfengine, rsync, more).

Different people also have different ways of building rpms. Some always proceed as root, for example, using /usr/src/redhat (which exists for that purpose, after all). However, in my own mind working as root is something to be avoided as much as possible because of the risk of unintended consequences when one makes a mistake. Some of you reading this HOWTO may be very uncomfortable working as root for this very reason. I therefore provide step by step instructions on how to turn the yum tarball into an rpm as *either* a user *or* as root.

Naturally, the rpm will have to be installed as root either way

Or at least, I'll provide them later. For the moment, I'm all tapped out and have to go anyway. I may not even return to this project for a few days. Told ya' it was a work in progress....

### 7.1 Creating a Yum RPM as Root

### 7.2 Creating a Yum RPM as a User

## 8 Installing Yum

Yum is installed (originally) more or less like any other rpm you add to an existing system. After downloading one of the rpm's above you simply:

```
rpm -Uvh yum.X.rpm
```

where X is, of course, the revision number you downloaded. Do this initially on the server(s) you wish to use as yum repositories and perhaps a single test client only. If this install fails, be sure that yum's dependencies are satisfied on your system. As of right now its dependency list reads something like:

```
rgb@lilith|T:167>rpm -qR yum
/bin/sh
/bin/sh
/bin/sh
/sbin/chkconfig
/sbin/service
/usr/bin/python
config(yum) = 1.97-20030522
python
rpm-python >= 4.2
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
```

on Red Hat 9. Version numbers will of course change, but this gives you an idea of what it will need in order to install. If you simply cannot get an install to work, try a different rpm or seek help from the yum list.

It is beyond the scope of this HOWTO to describe how to set up any of the servers that can be used as vehicles for a yum repository. Fortunately, documentation for setting up an FTP or HTTP server on a linux (or general Unix) server exists in abundance. Interested novices are referred to *The Linux Documentation Project* `http://www.tldp.org` website, where one can find HOWTO's (like this one), FAQ's, and outright books on doing nearly anything complex on a linux system. Your favorite technical bookstore is likely to have good books on at least the more important tools (such as apache).

For our purposes here, we will assume that you already have a functional HTTP or FTP server that you wish to make into a repository. It will need good bandwidth to all the clients you wish it to serve. It may need mechanisms to be in place to restrict access to the server to a local domain, or you may be comfortable permitting the server to be used by outsiders. It should be *very strictly controlled* in terms of access and security as it is an obvious target for crackers seeking to take over an entire network by slipping a malware rpm onto your repository for "automated update" to an entire network at once.

A double-barrelled shotgun behind the server room door, loaded with salt and tacks, is not out of the question.

With this given, we will now present instructions for setting up an rpm repository and preparing it so that yum clients can use it for all of their operations.

# 9 Yummifying your servers: yum-arch

## 9.1 Non technical explanation.

Basically all yum-arch does is digest the rpms that you have in a directory and create a header file for each rpm. That statement is *VERY HIGH LEVEL* and is not meant to diminish the importance of the yum-arch program. yum-arch is the cornerstone of yum. Without it you may as well stick with rpm.

## 9.2 Creating headers for an ftp server.

You will need to run yum-arch as root.

It is important that you create the yum headers in the correct directory. You could, if you wanted to, run yum-arch at the beginning of your repository and create one large header directory of all the rpm headers. You could, but you shouldn't.

The headers directory that you create will have a direct impact on what you will put in your yum.conf file so you will need to think about this carefully. You will have to point yum (via yum.conf) at the directory that has the headers directory in it or yum will not find the headers and not find the packages.

You must make sure that you set out your directory structure clearly and correctly. This just makes thing easier when something goes wrong. It also makes it easier to rsync your local mirror if you replicate the remote servers directory structure.

An example tree in /var/ftp/pub/ could look like this:

```
/9/
  /base/
```

```
          /athlon
          /i386
          /i586
          /i686
     /freshrpms
     /fedora
     /kde/
         /3.1.2
         /3.1.3
     /rawhide
     /updates/
             /athlon
             /i386
             /i586
             /i686
```

So if you ran

```
root@cartman>yum-arch /var/ftp/pub/9
```

yum-arch would create a 'header' directory in /var/ftp/pub/9 and place all the header files for the entire tree below /9/ in that directory.

The directory structure would look as follows:

```
   /9/
     /base/
          /athlon
          /i386
          /i586
          /i686
     /freshrpms
     /fedora
     /headers             # yum-arch creates the headers here.
     /kde/
         /3.1.2
         /3.1.3
     /rawhide
     /updates/
             /athlon
             /i386
             /i586
             /i686
```

That would work fine but it is not the cleanest way to use yum-arch. There are also issues with some of the newer functionality in yum like groups when all your headers are dumped in one directory.

So a solution would be to run yum-arch against a distinct sub-repository in the repository tree. An example of this is:

```
root@cartman>yum-arch /var/ftp/pub/9/base
root@cartman>yum-arch /var/ftp/pub/9/freshrpms
```

```
root@cartman>yum-arch /var/ftp/pub/9/fedora
               etc...
```

After yum-arch has digested all the rpms you will have a header directory in each sub-repository and your directory structure will look like this:

```
/9/
  /base/
       /athlon
       /headers
       /i386
       /i586
       /i686
  /freshrpms/
            /headers
  /fedora/
         /headers
  /kde/
      /3.1.2
      /3.1.3
      /headers
  /rawhide/
          /headers
  /updates/
          /athlon
          /headers
          /i386
          /i586
          /i686
```

Now that you have created the headers manually you will need to maintain your yum.conf file to point the client at the headers so you can start using yum.

## 9.3 Advanced Commands

yum-arch has a number of options that can be used to affect the way that the way that the headers are created.

```
root@cartman>yum-arch [options] directory
```

### 9.3.1 check dependencies and conflicts in tree (-d)

```
root@cartman>yum-arch -d directory
```

This command is really useful if you have a large number of rpms in a repository and you just want to create headers for the most recent rpms. yum-arch -d is very verbose, you will get a large amount of information of what yum-arch is doing so it might be a good idea to output this to a file as there can be a large number of conflicts and dependancies.

Here is an example of a conflict:

```
Checking deps 91 % completewarning: package samba-client = 3.0.0-5rc1 was already added, replacing with samba-client <= 3.0.0-5rc1
```

And here is an example of a failed dependancy:

```
depcheck: package xmms-devel needs xmms = 1.2.7
```

These messages are very useful and point to where problems might be occuring in your repository. Lot of people ignore these messages but then that is their choice. Messages are there for a reason.

### 9.3.2 Potential solutions for conflicts.

This is pretty simple. Delete the oldest rpms from your repository. This should really be done using rsync (as noted above). rsync, used to replicate a remote mirror locally, can be configured to delete local files that do not exist on the remote server.

### 9.3.3 Potential solutions for dependancy issues.

This may or may not be simple.

If the dependency problem is caused by the required package just not being available on any of the repositories you are mirroring, you will either have to look for the package at *rpmfind* `http://www.rpmfind.net` or (if you have sources or the data that needs to be packaged) build the rpm yourself. Building rpm's is discussed below.

**A word of warning**: Any packages that are not on the mirrors may be deleted from your local repository by rsync (if you use –delete to maintain a perfect mirror) so you will need to keep these site-local packages in a "local" repository. You have been warned.

If the dependency problem is more complex (and it may be *very* complex if you have two or more packages from different repositories whose builders used different naming/numbering conventions, or if you have "legacy", possibly proprietary or closed source packages that require old versions of libraries while the rest of your applications require modern libraries) then you will have to work much harder to resolve them. Indeed, in some cases no resolution will be possible – there are plenty of sites out there that are running "obsolete" linux distributions because one of their mission-critical applications requires the libc or other libraries intertwined with that particular distribution and all its binaries.

Fortunately, just as yum permits one to easily upgrade (so why would one ever need a complex repository tree?) it permits one to easily maintain a complex repository tree, so the solution in the worst cases may well be to maintain separate repository trees for non-conflicted distributions. Sure, it's ugly, but in the open source world (or for that matter the closed source world) one cannot *make* writers of important packages comply with a common set of rules, one can only hope that they do and deal with it when they don't.

### 9.3.4 make output verbose. (-v and -vv)

```
root@cartman>yum-arch -v directory
```

This command is very useful if you want to see what yum-arch is doing with every package in your repository. You will need to output this data to a file to be able to use it as there is a large amount of information produced. Here is an example of the verbose output:

```
ignoring older pkg: redhat/9/i386/openssh-askpass-gnome-3.5p1-6.9.i386.rpm
Digesting rpm - openssh-askpass-3.5p1-6.9.i386.rpm - 95/3202
Already found tuple: openssh-askpass i386

ignoring older pkg: redhat/9/i386/openssh-askpass-3.5p1-6.9.i386.rpm
Digesting rpm - openssh-3.5p1-6.9.i386.rpm - 96/3202
Already found tuple: openssh i386

ignoring older pkg: redhat/9/i386/openssh-3.5p1-6.9.i386.rpm
Digesting rpm - openssl096-0.9.6-17.i386.rpm - 97/3202
Digesting rpm - openssl-0.9.7a-5.i386.rpm - 98/3202
Digesting rpm - pam_smb-1.1.6-9.9.i386.rpm - 99/3202
Digesting rpm - php-4.2.2-17.2.i386.rpm - 100/3202
Digesting rpm - php-devel-4.2.2-17.2.i386.rpm - 101/3202
Digesting rpm - php-imap-4.2.2-17.2.i386.rpm - 102/3202
Digesting rpm - php-ldap-4.2.2-17.2.i386.rpm - 103/3202
Digesting rpm - php-manual-4.2.2-17.2.i386.rpm - 104/3202
```

How much info do you need? Once you are happy that yum-arch is working correctly and generating the correct headers you do not need the output to be verbose. yum-arch is usually run as a cron job so you are not really aware of the output generated. It is good to know that it is there and has helped me to explain some anomalies in the past.

### 9.3.5 do not generate headers. (-n)

```
root@cartman>yum-arch -n directory
```

When checking dependencies, one may not want to actually generate headers. Perhaps one has just added a new package and wants to see if it conflicts, but doesn't want there to be a window where the headers set is "broken" if there are indeed major conflicts. The -n flag permits the automatic generation of headers to be surpressed.

### 9.3.6 make output more quiet. (-q)

```
root@cartman>yum-arch -q directory
```

Sssshhhhhhh.

The -q flag does reduce the amount of information that yum-arch generates but yum-arch will never run totally silent. Any conflicts or redundant packages will be reported to the screen. Once you are happy that yum-arch is doing what you want it to do (ie. the second time you run it) you hould create a cron job that will run yum-arch daily. The -q flag is the best flag to use as yum-arch creates the headers and only tells you about the warnings.

### 9.3.7 check pkgs with gpg and md5 checksums (-c) cannot be used with -n (obviously)

```
root@cartman>yum-arch -c directory
```

Hmmmmmm. You are just looking for hassles with this command. I can see its uses but not all packages have an MD5 checksum nor do they have a gpg signature. So if you are security conscious and you are not sure about the packages that you have downloaded then this flag is for you.

**[THIS FLAG DOES NOT WORK FOR ME – jp]**

#### 9.3.8 compress headers using gzip. (-z)

```
root@cartman>yum-arch -z directory
```

....

#### 9.3.9 pay attention to symlinks (Default is to ignore symlinks). (-l)

```
root@cartman>yum-arch -l directory
```

.....

#### 9.3.10 ...all you really need to do

```
root@cartman>yum-arch directory
```

When you use yum-arch to create the headers for your repository, yum-arch will always produce these two lines at the end of the screen report (except -q):

```
Total: 3202              #Total packages read
Used: 1521               #Total headers created
```

That is all you really need to know. What are thee two lines telling me? Basically I need to clean up my repository as more than half of my packages are not having headers created for them. This is normally due to th fact that some of my packages are duplicated in the repository or there are conflicts. This is probably caused by the fact that I have the rawhide packages in my repository. Bleeding edge and all that.

## 10 Setting up the Yum Client

### 10.1 /etc/yum.conf

There are two types of sections in the yum.conf file: main and server. Main defines all the global configuration options. The Server section(s) define the entries for each server.

The Main section must exist for yum to do anything. Here is a fairly typical Main:

```
[main]
cachedir=/var/cache/yum
debuglevel=2
logfile=/var/log/yum.log
pkgpolicy=newest
distroverpkg=redhat-release
```

Here is the meaning of the various options (straight from "man yum.conf", in case you haven't read this yet). All options are of the form:

```
option=value
```

Note that *most* of the options can be left at the default and yum will function just fine.

- **cachedir:** Directory where yum should store its cache and db files.
- **debuglevel:** debug level. valid numbers are 0-10. default is 2.
- **errorlevel:** another debug level. valid numbers are 0-2. default is 2.
- **logfile:** full directory and file name for where yum should write its log file.
- **assumeyes:** 1 or 0 - tells yum whether or not to prompt you for confirmation of actions. Same as -y on the command line. Default to 0.
- **tolerant:** 1 or 0 - Tells yum to be tolerant of errors on the command line with regard to packages. For example: if you request to install foo, bar and baz and baz is installed; yum won't error out complaining that baz is already installed. same as -t on the command line. Default to 0(not tolerant)
- **pkgpolicy:** newest or last - Package sorting order. When a package is available from multiple servers, newest will install the most recent version of the package found. last will sort the servers alphabetically by serverid and install the version of the package found on the last server in the resulting list. If you don't understand the above then you're best left not including this option at all and letting the default (newest) occur.
- **exclude:** list of packages to exclude from updates or installs. This should be a space separated list. Filename globs *,?,., etc are allowed.
- **exactarch:** 1 or 0 - set to 1 to make yum update only update the architectures of packages that you have installed. i.e.– with this enabled yum will not install an i686 package to update an i386 package.
- **commands:** list of functional commands to run if no functional commands are specified on the command line. i.e: commands = update foo bar baz quux none of the short options (-y, -e, -d, etc) will be taken in the field.
- **distroverpkg:** package to use to determine the "version" of the distribution - can be any package you have installed - defaults to redhat-release.
- **diskspacecheck:** set this to 0 to disable the checking for sufficient diskspace before the rpm transaction is run. default is 1 (to perform the check)
- **retries:** Set the number of times any attempt to retrieve a file should retry before returning an error. Setting this to 0 makes it try forever. Default to 3.
- **kernelpkgnames:** list of package names that are kernels - this is really only here for the kernel updating portion - this should be removed out in 2.1 series.
- **installonlypkgs:** list of packages that should only ever be installed, never updated - kernels in particular fall into this category. Defaults to kernel, kernel-smp, kernel-bigmem, kernel-enterprise, kernel-debug, kernel-unsupported.

In addition, there must be at least one [server] section. Note well that "server" here does *not* refer to a single physical machine, nor does it necessarily refer to a single repository, a server id (this is made clear in the man page for yum.conf). As such it is an arbitrary but unique label attached to a set of (usually identical/mirrored) repositories that will in general be on different physical machines to provide a measure of fallback robustness in operation.

**Note well** that yum creates cache directories according to the label in the [server] tag. If the fallback repositories are not in fact mirrors and consistent with the primary repository, odd things can be expected to occur that are very likely not going to be what you intended to happen.

Server sections inherit global options from [main], but also have options of their own. Consider the following example:

```
[duke-base]
name=Linux@Home $releasever - $basearch - Base
baseurl=http://192.168.1.2/dulug/$releasever/$basearch/
        http://install.dulug.duke.edu/pub/dulug/base/$releasever/$basearch
gpgcheck=0

[duke-distrib]
name=Linux@Home Distributable add-on packages - $releasever $basearch
baseurl=http://192.168.1.2/dulug/$releasever/$basearch-distrib/
        http://install.dulug.duke.edu/pub/dulug/add-on/distrib/$releasever/$basearch
gpgcheck=0

[duke-nondistrib]
name=Linux@Home Nondistributable add-on packages - $releasever $basearch
baseurl=http://192.168.1.2/dulug/$releasever/$basearch-duke/
        install.dulug.duke.edu/pub/dulug/add-on/duke/$releasever/$basearch
        http://localhost:33333/pub/dulug/add-on/duke/$releasever/$basearchgpgcheck=0
gpgcheck=0

[home]
name=Linux@Home personal add-on packages - $releasever $basearch
baseurl=http://192.168.1.2/dulug/$releasever/$basearch-local/
gpgcheck=0

[ayo]
name=freshrpms
baseurl=http://ayo.freshrpms.net/redhat/9/i386/freshrpms/
gpgcheck=0
```

This example illustrates much of the power of yum. There are five distinct repositories indicated. The first, duke-base, is linux@duke's Red Hat 9 public repository. However, I use yum at home to keep my LAN up to date through a low-bandwidth DSL link. I have some ten computers that update nightly and don't want to have to move headers and rpm's through the link for each one. So I maintain a full mirror via rsync of this repository, and use it by preference rather than the repository at Duke.

Duke has two more collections of packages it provides in different repositories. One is open source/free add-ons to the base Red Hat distribution. This repository is open to the public, but less likely to be of use to people who aren't associated with Duke. I use some packages from this repository and so include in yum.conf, again first from a local mirror in my home LAN and from Duke through the DSL link only as a fallback.

The second repository contains packages that for one reason or another cannot be openly distributed. There are proprietary, site-licensed packages there, or packages with private data, for example. Although as a Duke faculty person I'm entitled to use these packages at home, only duke.edu addresses can access the repository. However, I connect via an ISP.

The [duke-nondistrib] entry thus tries to update from the home mirror (which I have to rsync via an ssh forwarded port). If this doesn't work it tries to access it directly (which might succeed on e.g. my laptop, if I've brought it into Duke). Finally, if neither of these work it tries localhost:33333, hoping that I've hotwired an ssh port forward at that address in case my laptop is not at home *or* at Duke, but I suddenly need access while I'm on the road.

The [home] entry is where I put those rpm's I build at home strictly for local/home distribution. This is surprisingly useful – a good way to move around certain configuration files, for example, and there are a *few* packages I find to install that aren't maintained at Duke at all.

The [ayo] entry is a public rpm repository that contains one or two libraries that I need that aren't at Duke and that I don't want to build locally.

As one can see, yum can be amazingly powerful and flexible in the way that it is configured to use repositories (and their fallback mirros) in layers, each layer providing a different set of packages. Most yum clients will need a much simpler yum.conf than this – in many cases one with a single server entry. However, systems managers who wish to install packages in layers from a common public repository and one or more private repositories will find that yum is designed to accomplish this in the most natural manner possible.

## 10.2 /etc/cron.daily/yum.cron

## 10.3 /var/cache/yum

## 10.4 Distributing the Configuration

### 10.4.1 Rebuilding the yum RPM with them ready to rock

### 10.4.2 Push tools

# 11 Using the Yum Client

The current command synopsis for yum is:

```
Usage:  yum [options] <update | upgrade | install | info | remove | list |
        clean | provides | search | check-update | groupinstall | groupupdate |
        grouplist >

    Options:
     -c [config file] - specify the config file to use
     -e [error level] - set the error logging level
     -d [debug level] - set the debugging level
```

```
-y answer yes to all questions
-t be tolerant about errors in package commands
-R [time in minutes] - set the max amount of time to randomly run in.
-C run from cache only - do not update the cache
--installroot=[path] - set the install root (default '/')
--version - output the version of yum
-h, --help this screen
```

(as of version 2.0.4).

## 11.1 Privileged Yum Commands The following commands alter the filesystem by installing, updating, deleting files. The first time yum is run, they will also download and cache all the header files in all the repositories in /etc/yum.conf, wich also requires root privileges. Thereafter the unprivileged commands will generally work for all users, but the commands in the subsections below will only work as root.

### 11.1.1 Installing Packages with Yum

```
# yum install package1 [package2 package3...]
```

Yum checks to see if package1 is already installed and is the latest version. If not, it downloads package1 {\em and all its dependency packages} (saving them in its cache directory) and installs them. Additional packages can be listed on the same command line. The packages can be specified with standard filesystem globs. Some examples:

```
# yum install jpilot
```

This will look for the jpilot package (which provides a lovely interface for palm pilots) and install it if it exists on any of the repositories in any of the yum repositories.

```
# yum install festival\*
```

This will install all the packages (e.g. festival and festival-devel) required to run the festival speech generation program or write software that does speech synthesis. Note the "\" required to escape the glob "*" character from the shell.

### 11.1.2 Updating Packages with Yum

```
# yum update package1 [package2 package3...]
```

Yum checks to see if package1 is installed and is the latest version. If not, it downloads package1 {\em and all its dependency packages} (saving them in its cache directory) and re-installs them (effectively upgrading them). Additional packages can be listed on the same command line. The packages can be specified with standard filesystem globs. Some examples:

```
# yum update
```

One of the most important and useful of yum commands, in some ways *the* command for which yum was invented. This command updates all the installed packages on your system to the latest version on your repository set. This enables you to keep your system current and up to date with a simple script, and to update any package(s) at any time should it be necessary.

```
# yum update jpilot
```

This will look for the jpilot package (which provides a lovely interface for palm pilots) and install it if it exists on any of the repositories in any of the yum repositories.

```
# yum update festival\*
```

This will update all the packages in the festival speech generation suite that happen to be installed on the system. Note the "\" required to escape the glob "*" character from the shell.

### 11.1.3 Deleting Packages with Yum

One should always exercise care when removing packages. This is because of dependencies - if you remove a package upon which other installed packages depend (to use a shared library, read a configuration file, even invoke a binary) then those packages are left "dangling" and will be left in a partly or completely dysfunctional state. Yum will try to help you avoid possibly devastating side effects associated with package removal, but as is always the case with something like this, it does require that the user think carefully when using it. It is therefore most unwise to use yum non-interactively when removing packages.

Since it is yum's basic design philosophy to never leave the system in an inconsistent/broken state (yum contains no support for

```
--force
```

-like options) if you ask yum to remove a package, it will insist on also removing all packages that depend on that package. If the package that you are removing is one upon which something like X itself or Apache depends, you may find yourself removing {\em an entire user interface} or {\em your entire webserver} as the package removal process is recursive through all dependencies. In many cases, this will *not* be what you want.

Let's understand this. In some cases you may have intended to remove only a top-level package and not realized that some other installed package depends on it. Clearly yum is doing you a service in these cases as it will keep you from removing a package that you didn't realize that you actually needed for something else. In other cases you might argue that you are only removing the package to replace it with another that also satisfies the dependence, so why does yum insist on removing all the dangling packages when you're about to reconnect them?

There are several reasons. For one, unless you have worked very hard and fully expanded the interconnected dependency tree and have done extensive testing of all the programs and tools *in* that tree, you don't *know* that the package you are dropping in will actually function as a perfect replacement. It is not enough to know that it provides a needed file or program or library – one has to work through the version dependence of the needed file or program or library, as these can and do change with time. New features are added, old features go away, two programs with the same name that serve the same general purpose take argument lists that differ in some crucial way.

Of course in some cases you are simply removing a package and replacing it with a more recent package with the same name. However, this is what the *update* command above does for you already as an atomic entity that never leaves the dependencies dangling, and in general you should use it instead, as it is likely to do a far more careful job of resolving the dependencies and ensuring that the updated package will still suffice.

There is one more place where this sort of remove-and-replace activity is likely to occur - when a package is obsoleted by another package. A package is obsoleted when it is removed entirely from an entire distribution/dependency tree and replace by a completely different package (different name, possibly different contents). This happens fairly regularly, if rarely, especially when the obsoleted package provides a configuration file that is shared by several tools. RPM's obsoletion process is very tricky, and can break things even when used correctly as it depends on all the packages in the dependency tree doing the right thing. Package are *often* obsoleted when a distribution changes its revision number, as that is the right time to manipulate entire branches of the tree with minimal impact.

The yum *upgrade* command listed below is the solution to the problem of obsoletion. It functions much like *update*, except that it manages the RPM obsoletes.

In any event, the command syntax for package removal is:

```
# yum remove package1 [package2 package3...]
```

As noted above, it removes package1 *and all packages in the dependency tree that depend on package1*, possibly irreversibly as far as configuration data is concerned. Be *certain* that the list of removed packages that it generates meets with your approval before proceeding. Additional packages can be listed on the same command line, and the packages can be specified with standard filesystem globs although this makes the problem of certifying the list it generates for removal even more difficult.

### 11.1.4 yum upgrade

Upgrading is the same as updating (and takes similar arguments) except that, as noted above, it also resolves and manages RPM package obsoletes, which remove core packages upon which many things depend and replace them (and as many of the dangling dependencies as possible) with a new, consistent branch in the dependency tree. This is *not* a completely safe thing to do in many cases because the overall replacement process can be very wide-reaching and have side effects that are difficult to fully explore in a testing process.

Note that under ordinary circumstances one should almost never encounter obsoletes unless you are actually upgrading from one distribution revision to another or are mixing packages from two different distributions revisions into your repository. For a variety of reasons, the latter is a really bad idea unless you love administrative pain. For a variety of reasons, it is often done anyway, and one of the things that tempts the use of

```
rpm --force
```

and consequently "breaks" the RPM database so that it becomes nearly impossible to safely resolve dependencies in the future. Yum upgrade at least gives you your best chance not to egregiously break something in this process without fair warning.

Even the legitimate purpose of doing a full revision upgrade is fraught with peril. For example, in revision upgrades the entire format of key configuration files in /etc might well change, and all tools and functions that depend on them might also need to change all the way down at the API or ABI level. It is again difficult to know that the RPMs for the entire upgraded tree are sufficiently carefully built that they can manage to *both* remove the old configuration files *and* preserve their contents and port the contents into the newly supported format, if possible. It is not at all unlikely that configuration data may be lost across an upgrade and that a system will therefore require a certain judicious amount of reconfiguration afterwards to regain full functionality.

For a specific example of this, consider the gradual replacement of the old Unix

`lpr`

printing system with the newer

`CUPS`

(Common Unix Printing System). Every aspect of printing configuration changes between the two, and it is nearly impossible to upgrade from one to the other without totally redoing the way printing is managed.

Because of these issues and yum's desire to function and do no harm in the process, it is likely the the

`yum upgrade`

function will be deprecated in the not horribly distant future. For that reason it is listed last in this HOWTO. In the meantime, it can certainly be a useful command and yum often does extraordinarily well at doing an upgrade, for all the dire warnings above. The author (rgb) of this HOWTO has used yum to upgrade Red Hat based systems on several occasions with complete satisfaction. He no longer does this - better practice is to develop e.g. a kickstart description of the systems in question that permits a full (re)install at any time into a perfectly functional state. This kickstart file is a much safer basis for upgrading the system to new distributions as they are released, as a full install eliminates the obsolescence process altogether, or at least forces one to confront precisely the relevant configuration issues as the emerge.

### 11.1.5 Yum Group Install/Update Commands

```
# yum group[install,update] group1 [group2 group3]
```

Yum has recently acquired the ability to perform the install, and update functions (as well as the list function, defined below) on *package groups* defined for a system, with otherwise similar arguments. This makes it relatively easy to install an entire block of software all at once, on systems that have package groups defined.

This can be very useful if one is using yum as a full scale system installation tool in its own right. For example, it is possible to put X onto a server originally installed without it by means of:

```
# yum groupinstall "X Window System" "X Software Development"
```

The

`groupupdate`

function is likely to be only infrequently required, as of course a

```
yum update
```

will update all packages on the system whether or not they were originally installed as a part of a group. One could imagine, perhaps, a circumstance where one wishes to update a particular group to current but *not* update the rest of your software installation to current, but I (rgb) have never encountered one.

## 11.2 Informational Yum Commands

### 11.2.1 Listing Available and Installed Packages with Yum

### 11.2.2 Getting information from packages with yum

### 11.2.3 Listing the Contents of a Package with Yum

### 11.2.4 Searching for a file in packages installed or uninstalled

# 12 Building RPMs for yum repositories

# 13 Security

# 14 Etc.

# 15 Further Information