Cross-Platform Multi-Instance Unix Software Packaging

package once — use everywhere!

Ralf S. Engelschall
rse@openpkg.org

The best way to predict the future is to invent it.
— Alan Kay

Only those who attempt the absurd can achieve the impossible.
— Unknown
Part I: Name Of The Game

What is the Problem?
Why Packaging at all?
Why Cross-Platform?

There are two types of people in this world, good and bad. The good sleep better, but the bad seem to enjoy the waking hours much more.

— Woody Allen
What is the Problem? (1)

- **Cross-Platform:**
  How to manage different Unix platforms without having to deal with different vendor facilities?

- **Trust:**
  How to trust any vendor unless their whole project workflow and results are public and transparent?

- **Organizational Separation:**
  How to achieve a clean responsibility separation on servers between System Administrators and Application Administrators/Developers?

- **Bleeding Edge:**
  How to use a software just a few hours after it was released by the vendor?

- **Package Variants:**
  How to deploy multiple variants (build-time options) of a software with an arbitrary vendor packaging facility?

- **Multiple Instances:**
  How to use staging installations without having to buy additional dedicated servers?

*Understanding a problem is knowing why it is hard to solve it, and why the most straightforward approaches won't work.* — Karl Popper
What is the Problem? (2)

- **Sane Build Environment:** How to build packages in a sane and well-defined environment?

- **Unprivileged Packaging:** How to build binary packages without write access to the target filesystem area?

- **Unprivileged Deployment:** How to use a software packaging facility in a fully unprivileged deployment environment?

- **Building from Source:** How to reproduce a software installation from pristine vendor sources directly on the end-user target machine?

- **Conciseness/Cleanness:** How to trust the resulting binary packages if the packaging itself is not already known to be maximum concise, clean and reviewable?

- **Safe Environment:** How to make sure that own solutions are future safe by not being too tied to a particular underlying operating system?

It’s not enough to be a great programmer; you have to find a great problem.
— Charles Simonyi
Why Packaging at all? (1)

**Reproducibility:**
Packaging allows to really reproduce the resulting software installation.

**Filesystem Intrusion:**
Packaging allows to exactly know what files form a piece of software. Later removal is possible without any residual files.

**Scalability:**
Packaging allows software deployment to be independent on the required number of deployments.

**Unification:**
Packaging unifies individual approaches across application vendors to simplify administration.

**Problem Focusing:**
Packaging allows to focus on the problem (deployment and configuration) instead of having to fight (again and again) against the porting and building of vendor software.

**Cost Reduction:**
Packaging reduces costs by no longer requiring experts for boring deployment tasks. Instead their expertise can be used for service improvements.

"Reuse an expert's code" is the right advice for most people. But it's useless advice for the experts writing the code in the first place. — Dan J. Bernstein
Why Packaging at all? (2)

- **Built-In Experience:** Packaging combines vendor applications with pre-configuration and packager knowledge to create optimum total result.

- **Knowledge Consolidation:** Packaging allows central consolidation of knowledge.

- **Patch Maintenance:** Packaging allows you to keep pristine vendor sources and patches separate without loosing seamless integration.

- **Annotations:** Packaging annotates vendor applications with useful meta information for administration.

- **Querying Information:** Packaging allows to reasonably query information about the application installation.

- **Safe Upgrade Path:** Packaging allows a guaranteed upgrade path for software during the whole life of a system.

- **Integrity Verification:** Packages can be signed and their content integrity can be verified.

The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man. — George Bernard Shaw
Why Cross-Platform? (1)
The Mountain Problem

- Different flavors of Unix operating systems have to be used and cannot be avoided.

- Differences in vendor supplied add-on applications:
  - Total number of applications.
  - Third-party application versions.
  - Used filesystem layout.
  - Particular chosen build-time options.
  - Amount of pre-configuration.

- Administrators have to know how to manage \textbf{n} different platforms.

There's a lesson to be learned from this but I'll be damned if I know what it is.
— Al Bundy
Why Cross-Platform? (2) The OpenPKG Solution

- Different flavors of Unix operating systems are still being used because cannot be avoided.
- Vendor supplied add-on applications are deinstalled or at least shadowed by OpenPKG.
- OpenPKG is a maximum independent layer on top of the operating system.
- All add-on applications are provided as cross-platform packages by OpenPKG.
- Administrators now just have to know how to manage 1 unified platform.

The software said it requires Solaris 9 or better, so I installed OpenPKG...
Part II: The Solution

The Solution: Overview
The Solution: Design Goals
Platform Availability
Platform Classification
Package Classification
Packaging Approaches

The solution of this problem is left as an exercise to the reader.
The Solution: Marketing Style

- OpenPKG — The Cross-Platform Multi-Instance Unix Software Packaging Facility.

- Much valued by IT decision makers and beloved by Unix system administrators, OpenPKG is the world leading instrument for deployment and maintenance of Open Source software when administration crosses Unix platform boundaries.

- The unique OpenPKG architecture leverages proven technologies like Red Hat Package Manager (RPM) and OSSP and GNU components to establish a unified software administration environment, independent of the underlying Unix operating system.

Software is like sex; it's better when it's free.
— Linus Torvalds
The Solution: Technology Style

- a cross-platform packaging facility for Unix software.
- based on a ported, cleaned up and extended version of the popular Red Hat Package Manager (RPM 4.2).
- a fully self-contained packaging facility which is maximum independent of underlying operating system.
- minimum intrusion during linkage into the underlying operating system (just 6 connection points).
- very complete, i.e., it currently provides already over 880 packaged applications.
- a mature technology now in production use since 4 years.
- freely available to anyone as Open Source under a MIT-style distribution license.
- releases are provided three times per year and the last two releases are fully covered with security updates.
The Solution: Design Goals

- Design Goal 1: Packaging at all (keywords: complexity, removability, reproducability, scalability)
- Design Goal 2: Cross-Platform (keywords: inherent constraints, flexibility, cost reduction)
- Design Goal 3: Multiple Instances (keywords: complexity, flexibility, utilization, evaluation, staging)
- Design Goal 4: Out-of-the-Box Configuration (keywords: minimum default, maximum usability, experience bundling)
- Design Goal 5: Accuracy & Conciseness (keywords: artwork, human friendliness, maintainability)
- Design Goal 6: Covering Essentials Only (keywords: “best of”, quality not quantity, major Unix flavors)
- Design Goal 7: Open Source Licensing (keywords: “free as in freedom, not as in free beer”)

Good design means less design. Design must serve users, not try to fool them.
— Dieter Rams, Chief Designer, BRAUN

I’d like to thank all the little people who helped make this possible, but I can’t, because I did it all myself.
— Herman Monster
The Solution: “Big Picture”

In the Classic approach, addon OS vendor packages plus manually installed software provide services.

![Diagram showing Classic Approach: OS1 Base, OS1 Addon, Manual, OS2 Base, OS2 Addon, Manual, HW1, HW2.]

In the OpenPKG approach, an OpenPKG Base instance extends the OS Base installation and dedicated OpenPKG instances provide services.

![Diagram showing OpenPKG Approach: OS1 Base, OpenPKG Addon, OS2 Base, OpenPKG Addon, HW1, HW2.]

The killer for system administration!

—we the unwilling, led by the unknowing, are doing the impossible.
—Larry Wall
Platform Availability

- OpenPKG is officially available for mainly 3 Unix platform technologies:
  - FreeBSD
  - GNU/Linux
  - Sun Solaris
- OpenPKG is officially available for 21 particular platform products (as of OpenPKG 2.3)
- For every release, all packages are built on all platforms.

- **FreeBSD**
  - FreeBSD 4.11 (iX86)
  - FreeBSD 5.3 (iX86)
  - FreeBSD 5.3 (SPARC64)
  - FreeBSD 5.3 (IA64)
  - FreeBSD 6.0 (iX86)

- **GNU/Linux**
  - Debian GNU/Linux 3.0 (iX86)
  - Debian GNU/Linux 3.1-PRE (iX86)
  - RedHat Enterprise Linux 3 (iX86)
  - Fedora Core 3 (iX86)
  - SuSE Enterprise Linux 9 (iX86)
  - SuSE Linux 9.2 (iX86)
  - Gentoo Linux 1.6.9 (iX86)
  - Mandrake Linux 10.1 (iX86)

- **Sun Solaris**
  - Sun Solaris 8 (SPARC64)
  - Sun Solaris 9 (iX86)
  - Sun Solaris 9 (SPARC64)
  - Sun Solaris 10 (iX86)
  - Sun Solaris 10 (SPARC)

- **Others**
  - NetBSD 2.0 (iX86)
  - HP HP-UX 11.11i (HPPA)
  - Apple Darwin 7.8 (PPC)
# Platform Classification

- OpenPKG platforms are classified into 5 categories:
  - deprecated
  - obsolete
  - supported
  - tentative
  - forecasted

- As the name implies, only “supported” platforms are really officially supported!

- Availability on “obsolete” platforms is still provided for convenience reasons only.

- Availability on “tentative” platforms is already provided for early adopter and testing reasons.

<table>
<thead>
<tr>
<th>Platform</th>
<th>5 Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>ix86-freebsd5.3</td>
<td>deprecated</td>
</tr>
<tr>
<td>ix86-freebsd4.11</td>
<td>obsolete</td>
</tr>
<tr>
<td>ix86-debian3.0</td>
<td>supported</td>
</tr>
<tr>
<td>ix86-fedora3</td>
<td>tentative</td>
</tr>
<tr>
<td>ix86-rhel3</td>
<td>forecasted</td>
</tr>
<tr>
<td>ix86-suse9.2</td>
<td></td>
</tr>
<tr>
<td>ix86-suse9</td>
<td></td>
</tr>
<tr>
<td>sparc64-solaris8</td>
<td></td>
</tr>
<tr>
<td>ix86-solaris9</td>
<td></td>
</tr>
<tr>
<td>sparc64-solaris9</td>
<td></td>
</tr>
<tr>
<td>ix86-solaris10</td>
<td></td>
</tr>
<tr>
<td>sparc64-solaris10</td>
<td></td>
</tr>
<tr>
<td>ix86-freebsd6.0</td>
<td></td>
</tr>
<tr>
<td>ia64-freebsd5.3</td>
<td></td>
</tr>
<tr>
<td>sparc64-freebsd5.3</td>
<td></td>
</tr>
<tr>
<td>ix86-gentoo1.6.9</td>
<td></td>
</tr>
<tr>
<td>ix86-debian3.1</td>
<td></td>
</tr>
<tr>
<td>ix86-mandrake10.1</td>
<td></td>
</tr>
<tr>
<td>ix86-netbsd2.0</td>
<td></td>
</tr>
<tr>
<td>ppc-darwin7.8.0</td>
<td></td>
</tr>
<tr>
<td>hppa-hpux11.11</td>
<td></td>
</tr>
<tr>
<td>ia64-freebsd5.3</td>
<td></td>
</tr>
<tr>
<td>sparc64-freebsd5.3</td>
<td></td>
</tr>
<tr>
<td>ix86-gentoo1.6.9</td>
<td></td>
</tr>
<tr>
<td>ix86-debian3.1</td>
<td></td>
</tr>
<tr>
<td>ix86-mandrake10.1</td>
<td></td>
</tr>
<tr>
<td>ix86-netbsd2.0</td>
<td></td>
</tr>
<tr>
<td>ppc-darwin7.8.0</td>
<td></td>
</tr>
<tr>
<td>hppa-hpux11.11</td>
<td></td>
</tr>
</tbody>
</table>

---

**Reporter:** What do you think of Western Civilization?

**Ghandhi:** I think, it would be a good idea.
Package Classification

- OpenPKG packages are classified into 5 categories:
  - CORE, BASE, PLUS
  - EVAL, JUNK

- Classification of a package depends on:
  - CORE, BASE: decision by principal architect.
  - PLUS: decision by principal architect and package status.
  - EVAL, JUNK: package status.

- The upper half of the “iceberg” (CORE, BASE and PLUS) make up the official releases.

- PLUS packages are going in and out as necessary during release engineering.

---

<table>
<thead>
<tr>
<th></th>
<th>CORE</th>
<th>BASE</th>
<th>PLUS</th>
<th>EVAL</th>
<th>JUNK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packaging Completed</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
</tr>
<tr>
<td>Build-Time Tests Successful</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>?</td>
<td>-</td>
</tr>
<tr>
<td>Run-Time Tests Successful</td>
<td>X</td>
<td>X</td>
<td>(X)</td>
<td>?</td>
<td>-</td>
</tr>
<tr>
<td>Release: Showstopper</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Release: Source Package</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Release: Binary Package</td>
<td>X</td>
<td>X</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Security Engineering</td>
<td>X</td>
<td>X</td>
<td>(X)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

"If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization." — Weinberg’s second law
Packaging Approaches: Source vs. Binary

- There are two fundamentally different approaches for packaging-based software distributions:
  - providing source packages containing the vendor sources plus instructions for automated build and installation.
  - providing binary packages containing the final installation files only.

- Most packaging facilities support both approaches (including RPM), although often not equally well.

- Both approaches have each their pros and cons, nevertheless all software distributions focus on one of them.

OpenPKG is focused on source packages because of the proofed success of reproducably building from pristine vendor sources.

In OpenPKG, binary packages are just an intermediate temporary result (or used for bootstrapping and emergency situations) only.

<table>
<thead>
<tr>
<th></th>
<th>source package</th>
<th>binary package</th>
</tr>
</thead>
<tbody>
<tr>
<td>distribution size</td>
<td>☺☺☺</td>
<td>☺☺☺</td>
</tr>
<tr>
<td>package size</td>
<td>☺</td>
<td>☺</td>
</tr>
<tr>
<td>package dependencies</td>
<td>☹</td>
<td>☺</td>
</tr>
<tr>
<td>installation reproducability</td>
<td>☹</td>
<td>☺☺☺</td>
</tr>
<tr>
<td>installation run-time stability</td>
<td>☺☺</td>
<td>☺</td>
</tr>
<tr>
<td>installation system alignment</td>
<td>☺☺☺</td>
<td>☺</td>
</tr>
<tr>
<td>installation time</td>
<td>☹</td>
<td>☺</td>
</tr>
</tbody>
</table>

Beware of programmers who carry screwdrivers.
— Leonard Brandwein
Part III: About Project

Project Roots
Project Roadmap
Engineering Phases
Who’s Who?

A distributed system is one on which I cannot get any work done, because a machine I have never heard of has crashed.

— Leslie Lamport
OpenPKG dates back in concept to 1992 when Ralf S. Engelschall (RSE) developed his Build’n’Play (BnP) and GenOPT at sd&m (sdm.de).

BnP was a Perl based build environment for easy installation of Unix software on FreeBSD and Sun Solaris.

GenOPT was a complex shell script which allowed to link the locally installed software into a global access layer.

When in November 2000 RSE went to Cable & Wireless (cw.com) the BnP/GenOPT approach was not sufficient and a more complete and integrated solution was aspired.

In-depth evaluation of major packaging facilities showed that none was able to fulfill all(!) requirements.

Fortunately, RPM proved to be the most balanced solution, because it covers at least 80% of every(!) requirement.

RPM was chosen, ported to more non-RedHat-Linux platforms and embedded into a elaborate bootstrapping procedure.

On top of this, the first dozen RPM packages were developed by converting the BnP Perl/sh scripts to RPM Bash scripts.

OpenPKG 0.9 was born!
OpenPKG RPM: PM Requirements

- The OpenPKG project had the following major requirements to the Package Manager:
  - The PM has to be maximum portable to all major Unix platforms and require the minimum on other software.
  - The PM has to cover the full life-cycle of a package, starting from tracking the vendor sources to the residue-free deinstallation of the installed package.
  - The PM has to be flexible enough to be easily extensible with OpenPKG extensions.
  - The PM has to be driven with a single all-in-one package specification and through a integrated command line interface.

- The OpenPKG project evaluated (in Nov. 2000) the following PM implementations:
  - FreeBSD 4.x Ports/pkg_xxx
  - Debian 2.2 dpkg/Apt
  - Sun Solaris 8 pkgxxx
  - RedHat RPM 4.0

- OpenPKG chose RedHat RPM because
  - it covered already 80% of all OpenPKG requirements.
  - the remaining 20% were added easily by OpenPKG.

- As of OpenPKG 2.3, the RPM 4.2.1 extensions are about
  - 9500 LoC shell extensions
  - 5000 LoC C patches
  - 450 LoC macro additions
  - 150 LoC CLI aliases

Engineering does not require science. Science helps a lot, but people built perfectly good brick walls long before they knew why cement works. — Alan Cox
About Project: The Roadmap

- As of April 2005, OpenPKG already went through 8 official releases since 2001.

- Release Engineering is performed within 4-6 weeks every 4 months in order to ship 3 releases per year.

- Security Engineering is performed constantly for the last 2 releases.

- OpenPKG-CURRENT is constantly updated on a bi-daily basis with the latest vendor versions.

<table>
<thead>
<tr>
<th>Date</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nov-2000</td>
<td>OpenPKG project kick-off</td>
</tr>
<tr>
<td>Apr-2001</td>
<td>OpenPKG 0.9, C&amp;W deployment</td>
</tr>
<tr>
<td>Jan-2002</td>
<td>OpenPKG 1.0</td>
</tr>
<tr>
<td>Mar-2002</td>
<td>feature: {s,m,r,n}{usr,grp}</td>
</tr>
<tr>
<td>Jun-2002</td>
<td>feature: sane build environment</td>
</tr>
<tr>
<td>Aug-2002</td>
<td>OpenPKG 1.1</td>
</tr>
<tr>
<td>Nov-2002</td>
<td>feature: RDF, openpkg-tool, FSL</td>
</tr>
<tr>
<td>Dec-2002</td>
<td>feature: %option</td>
</tr>
<tr>
<td>Jan-2003</td>
<td>OpenPKG 1.2</td>
</tr>
<tr>
<td>Apr-2003</td>
<td>feature: GCC 3.3, RC work-off</td>
</tr>
<tr>
<td>Aug-2003</td>
<td>OpenPKG 1.3</td>
</tr>
<tr>
<td>Oct-2003</td>
<td>feature: RPM 4.2.1, platform</td>
</tr>
<tr>
<td>Jan-2003</td>
<td>feature: UUID, %track/Class, tag</td>
</tr>
<tr>
<td>Feb-2004</td>
<td>OpenPKG 2.0</td>
</tr>
<tr>
<td>May-2004</td>
<td>feature: OpenPKG Tool Chain, gcc 3.4</td>
</tr>
<tr>
<td>Jun-2004</td>
<td>OpenPKG 2.1</td>
</tr>
<tr>
<td>Oct-2004</td>
<td>OpenPKG 2.2</td>
</tr>
<tr>
<td>Feb-2005</td>
<td>OpenPKG 2.3</td>
</tr>
<tr>
<td>Mar-2005</td>
<td>OpenPKG Foundation e.V., SpaceNet</td>
</tr>
<tr>
<td>Jun-2005</td>
<td>OpenPKG 2.4</td>
</tr>
<tr>
<td>Oct-2005</td>
<td>OpenPKG 2.5</td>
</tr>
<tr>
<td>Dec-2005</td>
<td>OpenPKG GmbH, OpenPKG Registry</td>
</tr>
<tr>
<td>Mar-2006</td>
<td>OpenPKG Websites 2.0</td>
</tr>
<tr>
<td>Jun-2006</td>
<td>OpenPKG 2.6</td>
</tr>
</tbody>
</table>

Some people have entirely too much **free time** on their hands. — Gene Spafford
About Project: Engineering Phases

- There are three types of recurring and overlapping phases in OpenPKG:
  - Development (Dev)
  - Release Engineering (RE)
  - Security Engineering (SE)

- Development Phase: implement new features, major changes, work-off packaging, ...

- Release Engineering Phase: fix building of packages, prepare release documents, ...

- Security Engineering Phase: ongoing effort to track security issues, backport and prepare patches, write security advisories, ...

Recursive, adj.; see Recursive.
Engineering Phases: Release Engineering

- Release Engineering is the recurring procedure where a new OpenPKG release is made.
- The frequency of 4 months is a balance between...
  - making the latest vendor software versions available for production environments.
  - providing a stable and consistent set of packages.
  - able to support risk free security updates for existing installations.
  - allow reproducible installations through fixated package versions.
  - having a limited amount of sponsored and contributed manpower and resources available.

- The Release Engineering steps mainly involve:
  - updating the OpenPKG build farm to the latest OS vendor versions/patchlevels.
  - fixing all CORE/BASE/PLUS packages to work on all supported platforms.
  - blessing EVAL class packages for PLUS class if they work on all platforms in order to increase release extend.
  - rolling the source and binary package distribution on all platforms for CORE/BASE/PLUS.
  - quality testing the packages.
  - updating documentation and publically publishing the results.

A new release is where old bad assumptions are replaced by new bad assumptions.
Engineering Phases: Security Engineering

- Security Engineering is an important task in OpenPKG because every release has a lifetime (usually 8 months).

- During the release life-time, existing installations are maintained with on-demand security updates.

- Deploying an OpenPKG security update is risk free, i.e., the user is guaranteed that no incompatible functional change or even new feature exists in any release update packages.

- The OpenPKG project achieves this by fully back-porting security fixes to the actually packaged vendor version. There is no simple vendor version upgrade made.

- The OpenPKG community is informed through public security advisories, summarizing the security issue and providing detailed information about affected releases and package versions.

- The OpenPKG project participates in closed vendor forums to get earliest possible notifications about security issues and to share own informations with other vendors.

- As a result of the ongoing OpenPKG security engineering process, the community gets security fixes as fast as possible.

The only secure computer is one that’s unplugged, locked in a safe, and buried 20 feet under the ground in a secret location... and I’m not even too sure about that one.

— Dennis Huges, FBI.
Who’s Who? (1)
Ralf S. Engelschall

Person Details:
- Name: Ralf S. Engelschall
- Born: November 17th, 1972
- Nationality: German
- Status: married, 2 children
- Profession: Computer Scientist
- Experience: 18 years of computing

Ralf S. Engelschall is the founder and principal architect of the OpenPKG project.

He is the author of about 90% of all OpenPKG packages.

Together with the OpenPKG Foundation he holds the copyright on OpenPKG.

He is also founder and president of the OpenPKG Foundation e.V.

His other well-known Open Source Software achievements:
- founder, principal architect and author of OSSP.
- co-founder and developer at OpenSSL.
- founder and author of Apache mod_ssl, author of Apache mod_rewrite, mod_dso and APACI.
- developer at FreeBSD.

A hacker does for love what others would not do for money.

Ralf S. Engelschall
rse@engelschall.com
rse.engelschall.com
Who’s Who? (2)
OpenPKG Foundation e.V.

- The social community around OpenPKG forms up in the OpenPKG Foundation e.V. http://www.openpkg.net/
- Excerpt from the Foundation constitution: “Intention of the OpenPKG Foundation e.V. is the ideational, financial, material and manned support of the Open Software Project OpenPKG.”
- The OpenPKG Foundation is a non-profit organisation, founded 2005 by Ralf S. Engelschall, Thomas Lotterer and OpenPKG developers.

The OpenPKG Foundation is established as an association under German law and regulated by a registered association constitution and companion bylaws following democratic rules.

**Teamwork** is essential:
There is always one you can **blame** it on.

When I was a boy I was told anybody can become **president**. I’m beginning to believe it...  — Clarence Darrow
Since 2005, the primary sponsors of OpenPKG are:

- **OpenPKG Foundation e.V.**
  http://www.openpkg.net/
  providing human resources and hardware resources.

- **SpaceNet AG**
  http://www.space.net/
  providing hosting resources and network resources.

If a **train station** is where trains stop, what is a **workstation**?
A supercomputer is a machine, that runs an endless loop in just 2 seconds.
OpenPKG RPM Crash-Course

- **Bootstrapping Instance:**
  ```bash
  $ sh openpkg-*.src.sh
  $ sh openpkg-*.*--*.sh
  ```

- **Installing Packages:**
  ```bash
  $ openpkg rpm -rebuild /
  foo-*.src.rpm
  # openpkg rpm -Uvh /
  foo-*.*--*.rpm
  ```

- **Starting/Stopping Services:**
  ```bash
  # openpkg rc foo stop start
  # openpkg rc foo status
  ```

- **Removing Packages:**
  ```bash
  # openpkg rpm -e foo
  ```

- **Removing Instance:**
  ```bash
  # openpkg rpm -e `openpkg rpm \
  -q --whatrequires openpkg`
  # openpkg rpm -e openpkg
  ```

- **Query Information:**
  ```bash
  $ openpkg rpm -qa
  $ openpkg rpm -qi foo
  $ openpkg rpm -qlv foo
  $ openpkg rpm -qf \
  /path/to/file
  $ openpkg rpm -qpi \
  foo-*.rpm
  $ openpkg rpm -qp \
  --requires foo-*.rpm
  ```

- **Verify Integrity:**
  ```bash
  # openpkg rpm -V foo
  # openpkg rpm -Va
  ```

- **Reading RPM Manual:**
  ```bash
  $ openpkg man rpm
  ```

---

Everybody falls the first time.  
It doesn't mean anything. 
— *The Matrix*
Build binary from source bootstrap package

```
$ TMPDIR=/var/tmp; export TMPDIR; cd $TMPDIR
$ ftp ftp.openpkg.org
Connected to ftp.openpkg.org.
Name (ftp.openpkg.org): anonymous
331 Anonymous login ok, send your email address as password.
Password: you@example.com
230- [...] Welcome to OpenPKG.org! [...] 
230 Anonymous access granted, restrictions apply.
ftp> bin
200 Type set to I.
ftp> cd release/2.5/SRC
ftp> get openpkg-2.5.0-2.5.0.src.sh
ftp> bye
221 Goodbye.
$ sh ./openpkg-2.5.0-2.5.0.src.sh --tag=opkg \
   --prefix=/usr/opkg --user=opkg --group=opkg
OpenPKG 2.5-RELEASE Source Bootstrap Package, version 2.5.0
Building for prefix /usr/opkg on current platform
++ extracting OpenPKG source distribution
++ building OpenPKG binary distribution
[...] 
$ ls -l openpkg-*
-rw-r--r-- 1 foo foo 18558976 Oct 20 10:20 openpkg-2.5.0-2.5.0.src.sh
-rw-r--r-- 1 foo foo 16997568 Oct 20 10:20 openpkg-2.5.0-2.5.0.src.rpm
-rw-r--r-- 1 foo foo 6230016 Oct 20 10:20 openpkg-2.5.0-2.5.0.i86-freebsd5.4-opkg.sh
-rw-r--r-- 1 foo foo  6230016 Oct 20 10:20 openpkg-2.5.0-2.5.0.i86-freebsd5.4-opkg.rpm
$ _
```

“The idea is to fall and miss the ground.”
— Douglas Adams,
A Hitchhiker's Guide to the galaxy.
Install binary bootstrap package to create instance

```
$ su -
Password: *****
# sh ./openpkg-2.5.0-2.5.0.ix86-freebsd5.4-opkg.sh
OpenPKG 2.5-RELEASE Binary Bootstrap Package, version 2.5.0
Built for prefix /tmp/openpkg on target platform ix86-freebsd5.4
++ hooking OpenPKG instance into system environment
++ creating OpenPKG instance root directory "/usr/opkg"
[...]
# exit
$ ls -l /usr/opkg
-rw-r--r--  1 opkg  opkg  911 Oct 20 10:20 README
drwxr-xr-x  6 opkg  opkg  512 Oct 20 10:20 RPM
drwxr-xr-x  2 opkg  opkg  512 Oct 20 10:20 bin
drwxr-xr-x  2 opkg  opkg  512 Oct 20 10:20 cgi
drwxr-xr-x  4 opkg  opkg  512 Oct 20 10:20 etc
drwxr-xr-x  3 opkg  opkg  512 Oct 20 10:20 include
drwxr-xr-x  2 opkg  opkg  512 Oct 20 10:20 info
drwxr-xr-x  3 opkg  opkg  512 Oct 20 10:20 lib
drwxr-xr-x  3 opkg  opkg  512 Oct 20 10:20 libexec
drwxr-xr-x 10 opkg  opkg  512 Oct 20 10:20 local
drwxr-xr-x 20 opkg  opkg  512 Oct 20 10:20 man
drwxr-xr-x  2 opkg  opkg  512 Oct 20 10:20 pub
drwxr-xr-x  2 opkg  opkg  512 Oct 20 10:20 sbin
drwxr-xr-x  2 opkg  opkg  512 Oct 20 10:20 share
drwxr-xr-x  2 opkg  opkg  512 Oct 20 10:20 var
$ /usr/opkg/bin/openpkg rpm -qa
openpkg-2.5.0-2.5.0
gpg-pubkey-63c4cb9f-3c591eda
$ _
```
Install OpenPKG package for GNU Bash (example)

```bash
$ /usr/opkg/bin/openpkg rpm --rebuild \
   ftp://ftp.openpkg.org/release/2.5/SRC/bash-3.0.16-2.5.0.src.rpm
Installing ftp://ftp.openpkg.org/release/2.5/SRC/bash-3.0.16-2.5.0.src.rpm
[...]
Wrote: /usr/opkg/RPM/PKG/bash-3.0.16-2.5.0.ix86-freebsd5.4-opkg.rpm
$ su -
# /usr/opkg/bin/openpkg rpm -Uvh /usr/opkg/RPM/PKG/bash-3.0.16-2.5.0.ix86-freebsd5.4-opkg.rpm
Preparing... #################################################### [100%]
# exit
$ /usr/opkg/bin/openpkg rpm -qlv bash
-rwxr-xr-x 1 opkg opkg    539068 Oct 20 10:20 /usr/opkg/bin/bash
drwxr-xr-x 2 opkg opkg     0 Oct 20 10:20 /etc/bash
-rw-r--r-- 1 opkg opkg    2756 Oct 20 10:20 /usr/opkg/etc/bash/profile
-rw-r--r-- 1 opkg opkg   342251 Oct 20 10:20 /usr/opkg/info/bash.info
-rw-r--r-- 1 opkg opkg   228383 Oct 20 10:20 /usr/opkg/man/man1/bash.1
$ /usr/opkg/bin/openpkg rpm -qi bash
Name:     bash
Version:  3.0.16
Release:  2.5.0
Group:    Shell
Class:    CORE
Distrib:  OpenPKG
License:  GPL
Packager: The OpenPKG Project
Vendor:   Free Software Foundation
[...]
$ _
```

Seek simplicity but distrust it.
— A. N. Whitehead
The lifecycle of a package is the most important part to understand in OpenPKG.

In OpenPKG, the lifecycle is an extended RPM package lifecycle because of extensions to RPM.

The lifecycle consists of overlapping steps performed by two parties:
- OpenPKG developers creating packages.
- OpenPKG administrators deploying packages.

The developer performs most of the administrator steps during build-time and run-time testing.

The administrator repeats some of the developer steps during building from source.

You can check out any time you like, but you can never leave.
— The Eagles, Hotel California
My hack: This universe. Just one little problem: core keeps dumping.
Part V: Developer Perspectives

Package Components
Package Specification
Package Building
Development: Version Tracking
Development: CVS Repository
Development: Build Farm

Computer science is no more about computers than astronomy is about telescopes.
— E. W. Dijkstra
Package Components

- Package Specification:
  - central OpenPKG RPM packaging information
    (name.spec)

- Vendor Sources:
  - vendor tarball
    (name-version.tar.gz)
  - vendor patches
    (name-version.patch)

- Extra Packaging Files
  - packager or third-party patches
    (name.patch[tag])
  - run-command scripts,
    FSL configurations, etc.
    (rc.name, fsl.name)
  - default configuration files
    (name.conf, ...)
  - ...

“UNIX is simple. It just takes a genius to understand its simplicity.”
— Dennis Ritchie
Package Specification

Every OpenPKG RPM package specification follows exactly the same structure and is strictly checked syntactically.

The section ordering is:
- macro defines
- package headers
- package options
- source references
- package dependencies
- package description
- version tracking
- build preparation
- configuration & building
- installation
- file determination
- cleanup
- deploy-time scripting

---

Bash (Bourne-Again Shell) is an sh-compatible command language interpreter that executes commands read from the standard input or from a file. Bash also incorporates useful features from the Korn and C shells (ksh and csh).

Politics is for the moment, an equation lasts eternity.  
— Albert Einstein
Package Specification (2)

In detail: Defines, Headers, Sources, Dependencies

```plaintext
# package version
#define V_base_real 3.0
#define V_base_comp 30
#define V_plvl_raw 16
#define V_plvl_pad 016

# package information
Name:  bash
Summary:  Bourne-Again Shell
URL:  http://cnswww.cns.cwru.edu/~chet/bash/bashtop.html
Vendor:  Free Software Foundation
Packager:  The OpenPKG Project
Distribution:  OpenPKG
Class:  CORE
Group:  Shell
License:  GPL
Version:  %{V_base_real}.%{V_plvl_raw}
Release:  2.5.0

# list of sources
Source0:  ftp://ftp.cwru.edu/pub/bash/bash-%{V_base_real}.tar.gz
Source1:  profile
Patch0:  bash.patch
...  ...

# build information
Prefix:  %{l_prefix}
BuildRoot:  %{l_buildroot}
BuildPreReq:  OpenPKG, openpkg >= 2.5.0
PreReq:  OpenPKG, openpkg >= 2.5.0
AutoReq:  no
```

What you see is all you get
— Brian Kernighan
Package Specification (3)

In detail: Description, Tracking, Preparation

```plaintext
%description
Bash (Bourne-Again Shell) is an sh-compatible command language interpreter
that executes commands read from the standard input or from a file. Bash
also incorporates useful features from the Korn and C shells (ksh and csh).
Bash is intended to be a conformant implementation of the IEEE POSIX Shell
and Tools specification (IEEE Working Group 1003.2).

%track
prog bash = {
    version = %{V_base_real}
    url = ftp://ftp.cwru.edu/pub/bash/
    regex = bash-(__VER__)\.tar\.gz
}
prog bash:patches = {
    version = %{V_base_comp}-%{V_plvl_pad}
    url = ftp://ftp.cwru.edu/pub/bash/
    regex = (bash-\d+\.\d+[a-z]+-patches)
    url = ftp://ftp.cwru.edu/pub/bash/__NEWVER__/_
    regex = bash\(\S+-\d+\)
}

%prep
# unpack and patch distribution
%setup -q -n bash-%{V_base_real}
%patch -p0 -P 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

# brand with OpenPKG release and fix patchlevel
{%{l_shtool} subst \n    -e 's;@1_openpkg_release@%;{l_openpkg_release};' \ version.c
%
{l_shtool} subst \n    -e 's;(PATCHLEVEL) 0;1 %{V_plvl_raw};' \ patchlevel.h
```

Beware of bugs in the above code;
I have only proved it correct, not tried it.
— D.E. Knuth
Package Specification (4)

In detail: Configuration and Building

```bash
# build
  # configure package
  ( # force disabled wide-character support
    echo "ac_cv_header_wchar_h=no"
    echo "ac_cv_header_wctype_h=no"
    echo "ac_cv_func_mbsrtowcs=no"
    # force disabled internationalization support
    echo "ac_cv_header_libintl_h=no"
    echo "ac_cv_func_gettext=no"
    echo "ac_cv_func_textdomain=no"
    echo "ac_cv_func_bindtextdomain=no"
    echo "ac_cv_lib_intl_bindtextdomain=no"
  ) >config.cache
CC="%{l_cc}" \
CFLAGS=""%{l_cflags -O}" \
./configure \
  --cache-file=./config.cache \
  --prefix=%{l_prefix} \
  --disable-multibyte \
  --enable-debugger \
  --without-gnu_MALLOC \
  --without-curses \
  --disable-nls
{l_shtool} subst \
  -e 's;^\(#define.*SYS_PROFILE[^]*\).*;l "${l_prefix}/etc/bash/profile";' \
  pathnames.h
{l_shtool} subst \
  -e 's;/etc/profile;${l_prefix}/etc/bash/profile;' \
  doc/bash.1

# build package
{l_make} %{l_mflags}
```
In detail: Installation, File Determination, Cleanup

%install
    # install package
    rm -rf $RPM_BUILD_ROOT
    %{l_make} %{l_mflags} install \n        prefix=$RPM_BUILD_ROOT%{l_prefix}
    # strip down installation
    rm -f $RPM_BUILD_ROOT%{l_prefix}/info/dir
    rm -f $RPM_BUILD_ROOT%{l_prefix}/man/man1/bashbug.1
    rm -f $RPM_BUILD_ROOT%{l_prefix}/bin/bashbug
    strip $RPM_BUILD_ROOT%{l_prefix}/bin/bash
    # install global configuration
    %{l_shtool} mkdir -f -p -m 755 \n        $RPM_BUILD_ROOT%{l_prefix}/etc/bash
    %{l_shtool} install -c -m 644 %{l_value -s -a} \n        %{SOURCE profile} $RPM_BUILD_ROOT%{l_prefix}/etc/bash/
    # determine installation files
    %{l_rpmtool} files -v -ofiles -r$RPM_BUILD_ROOT \n        %{l_files_std} \n        'conf %{l_prefix}/etc/bash/profile'
%files -f files
%clean
    rm -rf $RPM_BUILD_ROOT

The number of UNIX installations has grown to 10, with more expected.
## In detail: Post-Installation Processing

```bash
%post
  if [ ".$1" = .1 ]; then
    # display note about login shell prerequisite
    if [ -f /etc/shells ]; then
      if [ ".`grep $RPM_INSTALL_PREFIX/bin/bash /etc/shells`" = . ]; then
        ( echo "Hint: To use "$RPM_INSTALL_PREFIX/bin/bash as the login"
          echo "shell for users, please add this path to /etc/shells."
        ) | %{l_rpmtool} msg -b -t notice
      fi
    fi
  fi
fi
```

---

I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.

— C.A.R. Hoare
Package Building:
RPM Control/ Data Flow

Patient: Doctor, it hurts when I do this!
Doctor: Well, then don't do it.
Development: Version Tracking

- OpenPKG RPM supports a custom section %track which contains vcheck(1) configurations.
- A vcheck(1) configuration is:
  - last known version
  - URL where the versions are referenced
  - regular expression how the versions can be extracted from the text under the URL
- All 800 OpenPKG packages contain a %track section for checking all external source files of a package.
- On a bi-daily basis all %track sections are executed and a report sent to the OpenPKG developers.
- See also: openpkg-dev@openpkg.org

---

Life is just a beta-version. Don't expect it to be bug-free.
Development: CVS Repository

- All sources of OpenPKG are stored in a central CVS based repository system.
- Every “Commit” to the repository is real-time tracked both with detailed reports via Email and on-line via CVSTrac.
- Every OpenPKG release is an own “branch” in the repository.
- See also: http://cvs.openpkg.org/openpkg-cvs@openpkg.org

Murphy’s Law is recursive: Washing your car to make it rain doesn't work.
Modified files:
- opengpg-src/rsync
- rc.rsync
- rsync.spec

Log:
disable rsync daemon by default because in 90% of all cases the way rsync is used.

Summary:

Revision Changes Path
1.21 +1 -1 opengpg-src/rsync/rc.rsync
1.59 +1 -1 opengpg-src/rsync/rsync.spec

patch -p0 <<<`
Index: opengpg-src/rsync/rc.rsync
-------
cvs diff -u -r1.21 -r1.21 rc.rsync
--- opengpg-src/rsync/rc.rsync 7 Aug 2003 06:51:05
@@ -5,7 +5,7 @@

  #config
  rsync_enable="$openpgp_ro_def"
  - rsync_daemon="yes"
  + rsync_daemon="no"
  rsync_flags=""
  rsync_bind="127.0.0.1"
  rsync_port="973"

---

patch -p0 <<<`
Index: opengpg-src/rsync/rsync.spec
-------
cvs diff -u -r1.59 -r1.59 rsync.spec
--- opengpg-src/rsync/rsync.spec 7 Feb 2004 17:59:30
@@ -34,7 +34,7 @@

  Group: Filesystem
  License: GPL
  Version: 2.5.0
  -Release: 20040207
  +Release: 20040324

* list of sources
  Source0: http://rsync.samba.org/ftp/rsync/rsync-live/
Development: Build Farm

- OpenPKG packages are constantly tested on a large set of different platforms.
- For this a “build farm” is used (provided by the OpenPKG Foundation e.V.), consisting of machines which constantly fetch the latest OpenPKG-CURRENT and try to build changed packages.
- The result is a status page on the website which shows the latest status of each package on each platform.
- The developers watch this status page to see where something has to be fixed.
- See also: http://www.openpkg.org/status.cgi

The goal of **science** is to build better **mousetraps**. The goal of **nature** is to build better **mice**.
Part VI: Some Gory Details

The “Bootstrap” (Package)
Run-Command Facility (RC)
OSSP fsl (Faking Syslog Library)

A diplomat is someone who can tell you to go to hell in such a way that you will look forward to the trip.
OpenPKG technically consists of the essential “openpkg” RPM package plus 880 other RPM packages based on it.

The “openpkg” package is called “the bootstrap” because it is
- both a regular RPM package containing the RPM framework
- and an elaborate bootstrapping procedure able to install itself with itself from scratch.

This way OpenPKG RPM is 100% packaged by itself and especially is able to upgrade its RPM framework with itself.

The bootstrapping works by...
- emulating a minimum functional subset of RPM with a shell script.
- building and installing the “openpkg” package content with the RPM emulation content into a temporary area.
- faking the rebuild and in-place re-installation of the “openpkg” package with the RPM from the temporary area in order to record RPM into its own RPM database.
- rolling a bootstrapping binary shell script and binary RPM package from the temporary area.

All the good things you want to do in your life have to be started in the next few hours, days or weeks.
— Tom DeMarco
Run-Command Facility (1)
Overview

- OpenPKG provides a flexible and integrated Run-Command (RC) facility.
- The OpenPKG RC facility is...
  - based on ideas from the NetBSD 1.6 and FreeBSD 5 RC facility (no run-levels, rc.d/ directory, dependencies, shared RC shell functions, rc.conf functionality, etc).
  - designed with a RPM-style script sectioning syntax (e.g. %start) and an all-in-one specification approach for seamless integration into the RPM scope.
  - integrates both startup/shutdown (boot!) and periodic (cron!) run-command functionality.
- A run-command script in OpenPKG RPM and RC is always a GNU Bash script, independent of the underlying platform.
- As a result, for a particular packaged application...
  - the OpenPKG RPM package specification covers the build-time and install-time run-commands.
  - the OpenPKG RC package specification covers the run-time run-commands.
- The OpenPKG RC facility consists of:
  - prefix/etc/rc
  - prefix/etc/rc.func
  - prefix/etc/rc.conf
  - prefix/etc/rc.d/rc.package

To me Vi is Zen. To use Vi is to practice Zen. Every command is a Koan. Profound to the user, unintelligible to the uninitiated. You discover truth everytime you use it. — Achim Bohnet
Command Line Interface:
```bash
# openpkg rc package command
```

The `package` argument is
- either `foo` (particular package).
- or `all` (all packages at once).

The `command` argument is an arbitrary command corresponding to a "`%command`" section in `rc.package`.

The following commands are well-known and implemented by all packages with `rc.package`:
- `status`    start    stop

Other well-known sections:
- `restart`    `reload`
- `quarterly`  `hourly`  `daily`
- `weekly`     `monthly`

Two special sections exist:
- `%config`: contains defaults for configuration variables which can be overridden from `rc.conf`
- `%common`: contains run-commands common to all other sections (except `%config`)

Running `prefix/bin/openpkg rc foo start` runs a GNU Bash script assembled from
- `%config` sections from all `prefix/etc/rc.d/rc.*`
- sourcing of `prefix/etc/rc.conf`
- `%common` section from `prefix/etc/rc.d/rc.foo`
- `%start` section from `prefix/etc/rc.d/rc.foo`
OSSP fsl (Faking Syslog Library)

- An inherent design goal of OpenPKG is to support multiple instances.

- Major problems with multiple installations of the same application are
  - the listening to the network address/port.
  - the logging via the central syslog(3) facility.

- Conflicts on network listening most of the time can be solved easily by just re-configuring the application.

- Syslog(3) usage in multiple installations of the same application usually results in merged logfile entries in the central logfiles.

- OpenPKG solves the syslog(3) problem with OSSP fsl, a faking syslog(3) library.

- OSSP fsl emulates the syslog(3) API but instead of sending the log message to syslogd(8) it is sending it through a tree of chained channels.

- The tree of chained channels can be configured individually for each application through pattern matching in prefix/etc/fsl/fsl.package.

- OpenPKG by default links all applications using syslog(3) against OSSP fsl and directs their log messages to logfiles staying inside their OpenPKG instance (usually prefix/var/package/package.log)
Part VII: Finish

More about OpenPKG...

The Apache Group: a collection of talented individuals who are trying to perfect the art of never finishing something.

— Rob Hartill
More about OpenPKG...

- The Website: http://www.openpkg.org/
- The RSYNC Server: rsync://rsync.openpkg.org/
- The CVS Server: http://cvs.openpkg.org/
- The OpenPGP Key Server: http://pgp.openpkg.org/hkp://pgp.openpkg.org/
- The Community Mailing Lists: openpkg-announce@openpkg.org openpkg-users@openpkg.org openpkg-dev@openpkg.org openpkg-cvs@openpkg.org

---

I have made this longer than usual because I lack the time to make it shorter. — Blaise Pascal