RPM based Software Maintenance
on
DESY Unix Systems

Stephan Wiesand
DESY -DV-
January 27th, 2004

Warning

The term technical is an exact match today

Agenda

- brief recap of the /opt/products scheme
  - one year after
- package flavours
  - and their contents and purpose
- building packages
  - best practices
  - DOs and DON'Ts
- repository structure
- managing installations and repositories
  - tools developed during past 12 months

The `/opt/products` Scheme

- software products install into **`/opt/products`**:
  - the home of perl is **`/opt/products/perl`**
  - the home of perl version 5.8.2 is **`/opt/products/perl/5.8.2`**
    - this is the content of the RPM package **`perl-5.8.2`**
  - there may be a link into a bin directory in the user's path:
    - `/opt/products/bin/perl -> /opt/products/perl/5.8.2/bin/perl`
    - this is contained in the RPM package **`perl-default-5.8.2`**
- **`perl-5.8.2`** can be installed normally
  - or with **`rpm --justdb`**, and replaced by a single symlink
- **`perl-default-5.8.2`** can be installed normally only

The Reference Installation

- if **`/opt/products/perl/5.8.2`** is a symlink, it points to
  - **`/afs/cell/@sys/products/perl/5.8.2/`**
- to unload the content of **`perl-5.8.2`** there:
  - (temporarily) create a symlink into AFS on some system
    - **`/opt/products -> /afs/.cell/@sys/products`**
  - install the package
  - release all volumes that changed
- if you also install **`perl-default-5.8.2`**, a client can use all products through just one link
  - **`/opt/products -> /afs/cell/@sys/products`**
  - needs no maintenance at all, but no local packages/variations

The RPM Database

- our products do NOT use the same DB as system packages
  - the single link variant wouldn't work well (no DB available)
  - mixed clients need information about reference installation
  - some target platforms have no native rpm
  - => best location is below **`/opt/products`**
- chose **`/opt/products/RPMDB`**
- always use **`rpm --dbpath /opt/products/RPMDB`**
  - the **`prpm`** command does just that
    - but like most wrappers, it works for simple cases only
    - cannot deal with quoted arguments

Closing the Dependency Chain

- **`perl-5.8.2`** requires shared system libs, like libc.so.6
  - provided by the system packages
  - but our RPMDB doesn't know
- solution: package **`glue`** (and **`glue-devel`**)
  - **always** installed with **`--justdb`**, contains no real files
  - provides what our packages need
  - verifying glue will check that the system keeps its promises
  - straightforward on linux, but solaris package system is dumb
    - may have to verify actual existence of files, ... to be done

Automatic Dependency Detection

- at end of package build process, rpm calls some scripts
  - **`find-requires`** and **`find-provides`**
    - arguments: all files in package
    - returned:
      - requirements: shared libs, and script processors
      - capabilities: shared libs (a package provides all its files)
  - modified **`find-requires`** from prpm package
    - works on Linux and Solaris, perl, much faster than default
    - if a file depends on a shared lib in /opt/products, does NOT process this dependency
      - standard version would require **`libcrypto.so.0.9.6`**
      - we require **`/opt/products/openssl/0.9.6l/lib/libcrypto.so.0.9.6`**

Dependencies and Shared Libs

- our products shall not be broken by changes of shared libraries from other packages
  - burn an RPATH into all our executables for all libraries provided by our products
    - often the hardest part of building a proper package
  - yes, this means we may have to rebuild more often
  - but it also means new versions of products can be installed without breaking anything
  - dependencies allow easy identification of products to rebuild
  - dependencies guarantee necessary packages are installed
  - there were incompatible changes within openssl 0.9.6.x

Package Flavours

- we already know three different package types:
  - **glue** packages close dependency chain to system side
  - **base** packages
    - **perl-5.8.2** contains **/opt/products/perl/5.8.2**
      - can install **5.8.2** and **5.8.0**, but only one release of each
  - **-default** packages
    - **perl-default-5.8.2** contains **/opt/products/bin/perl**
      - only a single version of perl-default can be installed at a time
    - but how about **/opt/products/bin/perl582** ?
  - => **-alias** packages
    - multiple **perl-alias** versions may be installed (must not clash)

More Package Flavours

- how about perl modules ?
  - look like a base package, but have no own home directory
    - install into **/opt/products/perl/5.8.2/lib/...**
      - which cannot be replaced by a single symlink :-(
- => child packages
  - naming convention:
    - ***parentname_parentversion-childname***
      - **perl_5.8.0-URI**
      - **perl_5.8.2-URI**
      - **mozilla_1.6-plugins**
  - have their own version and release
  - major complication !

A Last Package Flavour

- what if a file must be allowed to change without updating ?
  - example: configuration files (**pine.conf**)
  - step 1: separate the file from the main package
    - could be a child package, or an own base package
    - in case of pine currently: base package **pineconfig**
    - pine reads **/opt/products/etc/pine/pine.conf**
      - a link coming from **pineconfig-default** (of course)
    - but what if we want to maintain it manually, or with cfengine ?
      - then it should be located on the client's local filesystem
  - => **-local** package **pineconfig-local**
    - **/opt/products/etc/pine -> /etc/opt/products/pine**

Package Flavours Summary: Base Packages

- ***name-version*** contains **/opt/products/*name/version***
  - if the RPM DB has an entry for this package
    - it is installed locally if and only if this directory exists
    - it is installed as link if and only if this is a symbolic link
    - otherwise, it must be installed with --justdb
  - must not contain any files outside **.../name/version**
- an arbitrary number of children may exist
- there may exist **-alias, -default, -local** packages
  - with identical version and release as the base package
- multiple versions can be installed
- only a single release of each version may be installed

Package Flavours Summary: -default Packages

- can be installed locally only
- contain links into home directory of their base package
  - anywhere under **/opt/products**
    - with the exception of base package home directories
    - ok: **/opt/products/cernlib/pro -> 2003**
      - /opt/products/cernlib is always a local directory
    - bad: **/opt/products/cernlib/2003/something -> anything**
- always depend on their base package
  - cannot be installed without them
  - share version and release of base package
- only a single version and release can be installed
  - there is only one **/opt/products/bin/perl**

Package Flavours Summary: -alias Packages

- share almost all characteristics of **-default** packages
  - local installation only
  - depend on their base package
  - share version and release of their base package
  - contain links into their base packages home directory
  - from anywhere in /opt/products except any base home
- exception: several versions can be installed at any time
  - this makes sense:
    - **maple-alias-7**
      - **/opt/products/bin/maple7 -> /opt/products/maple/7/bin/maple**
    - **maple-alias-8**
      - **/opt/products/bin/maple8 -> /opt/products/maple/8/bin/maple**

Package Flavours Summary: -local Packages

- are exactly like **-default** packages
  - local installation of a single version only
  - depend on their base package
  - share version and release of their base package
  - contain links into their base packages home directory
  - from anywhere in /opt/products except any base home
- but links point into the client's local filesystem
  - **apache-config** contains a sample configuration
  - **apache-config-default** makes apache use it
    - **/opt/products/etc/apache -> /opt/products/apache-config/etc**
  - **apache-config-local** makes apache use a local configuration
    - **/opt/products/etc/apache -> /etc/opt/products/apache**

Package Flavours Summary: Child Packages

- depend on their parent package (which must be of base type)
- name starts with ***parentname_parentversion-***
- have their own independent version and release
- only a single version can be installed
  - **perl_5.8.0-URI-1.2.4** and **perl_5.8.0-URI-1.2.5** clash
  - **perl_5.8.0-URI-1.2.5** and **perl_5.8.2-URI-1.2.5** don't
- may be installed locally if and only if their parent is
- are installed as link if and only if
  - their parent is installed as link, and
  - they are installed on the reference installation
- may have **-default, -alias, -local** packages

Building Packages

- this is not an RPM tutorial
  - covered in previous talks
  - basics are quite simple
    - just copy and modify an existing package
    - hundreds of examples available
    - excellent docs available (Maximum RPM, even online)
- it is a list of
  - best practices after one year of experience
  - common traps and pitfalls to avoid
  - tips & tricks not found in the RPM docs

Getting Started

- any member of group sysprog can install (no)source RPMs
  - **rpm -ivh /packages/SRPMS/gcc/gcc-3.3.2-1.nosrc.rpm**
    - **/usr/src/packages/SPECS/gcc-3.3.2-1.spec**
    - sources and patches in **/usr/src/packages/SOURCES**
- a source rpm contains the spec and all other input files
  - **Source0: gcc-3.3.2.tar.bz2**
  - **Patch0: my-gcc-patch.diff**
- a nosource rpm explicitly excludes some input files
  - **NoSource: 0**
  - the .nosrc.rpm still contains the patch, but not the tarball
  - .nosrc.rpm is small enough to be stored forever

SPEC Basics: Naming Convention, Input Files

- SPECs can have arbitrary names, but please use
  - ***name-version-release*.spec**
- declare ALL sources and patches, but exclude huge files
  - make sure excluded files are in the repository
    - make it a habit to access them through links only
  - record the MD5 checksum of excluded files in the package
    - **%(...)** macros substitue command output:

```
Source0:  gcc-3.3.2.tar.bz2
NoSource: 0

%description
blabla...
Source MD5 sum: %(md5sum %{SOURCE0} | sed s@%{_sourcedir}/@@)
```

SPEC Basics: Meta Data

- pay attention to the data in the preamble
  - packager, copyright, summmary, group, ...
  - requirements, provides, conflicts, build requirements
  - provide a decent description
- make use of the changelog (at the end of the file)
  - what is the reason for the new version/release ? security ?
  - any build changes ? libs, compiler, tweaks, patches, ... ?
  - keep previous entries if it makes sense
    - version 3.10.02 derived from 3.10.01 is NOT an "initial version" !
      - at least if all you changed is the version and release numbers

SPEC Basics: Macros

- make generous use of spec file macros
  - greatly improves reusability
- most useful:
  - `%{name}, %{version}, %{release}`
  - `%{SOURCE0}, %{SOURCE1}, ..., %{PATCH0}, ...`
  - `%{buildroot}`
  - `%(shell command)`
  - `%ifarch, %ifos`
- sometimes useful:
  - `%{_builddir}, %{_sourcedir}`

SPEC: Build Root

- always use a build root (declare in preamble):
  - `BuildRoot: %{_builddir}/%{name}-buildroot`
- implicitly prepended to all path in files section
- package still unpacks files to right destination
- useful for keeping new files separate from system
  - in particular when building -default and -alias packages
  - actually collected from `%{buildroot}/opt/products/...`:

```
%files default
/opt/products/bin/*
/opt/products/man/man1/*
```

Multiple RPMs from a single SPEC

- create the base and -default packages from same SPEC
- `%package default`
  - declares package `%{name}-default-%{version}-%{release}`
  - must be followed by specific `Summary:` and `Group:` tags
  - optionally, own `Requires:` and other tags
  - followed by mandatory `%description default`
- sections `%prep, %build, %install` are all shared
  - as is the build root (unfortunately)
- `%files default`
  - separate files section for the -default package

-default & Build Root: Procedure 1

- the lazy way:
  - install into /opt/products
  - create default directories & links in build root
  - move installed software into build root
  - disadvantage: what if this version is already installed ?
    - check in prep section, run "false" if it is

```
%build
./configure --prefix=/opt/products/%{name}/%{version}
make

%install
make install
mkdir -p %{buildroot}/opt/products/bin
ln -s /opt/products/%{name}/%{version}/bin/*  %{buildroot}/opt/products/bin/

mkdir -p %{buildroot}/opt/products/%{name}
mv /opt/products/%{name}/%{version} %{buildroot}/opt/products/%{name}
```

-default & Build Root: Procedure 2

- if the software supports DESTDIR installs (more & more)
  - install into build root right away, leaving /opt/products alone
    - somewhat more work
    - use shell loops and black magic (build shell is bash)
      - **${var##*prefix*}** expands to $var with ***prefix*** removed

```
%build
./configure --prefix=/opt/products/%{name}/%{version}
make

%install
make install DESTDIR=%{buildroot}

mkdir -p %{buildroot}/opt/products/bin

for i in %{buildroot}/opt/products/%{name}/%{version}/bin/*; do
   ln -s ${i##%{buildroot}} %{buildroot}/opt/products/bin/
done
```

-default & -alias & Build Root

- our file sections so far are pretty simple
  - base package: **/opt/products/%{name}/{version}**
  - default package: **/opt/products/bin/* ...**
- more complicated if **-default** and **-alias**
  - solution: use **%files -f *packlistfile***

```
%install
...
for i in %{buildroot}/opt/products/%{name}/%{version}/bin/*; do
   ln -s ${i##%{buildroot}} %{buildroot}/opt/products/bin/
   echo ${i##%{buildroot}} >> /tmp/files-for-default.txt
done
ln -s /opt/products/%{name}/%{version}/bin/%{name}%{version} \
   %{buildroot}/opt/products/bin/

%files default -f /tmp/files-for-default.txt

%files alias
/opt/products/bin/%{name}%{version}
```

-default & -local

- mutually exclusive
  - make it explicit with a **Conflicts:** tag
- should provide a common virtual package
  - required by base package
- building both in single spec is impossible
  - build root is global
  - -default and -local share files
    - not allowed even if they were identical
    - and in our case, they're links with different destinations
- => **-local** better built in separate spec (usually trivial)

Build Root & Dependencies

- problem if package requires its own files (libs, interpreters)
  - when dependencies are calculated, the're in the build root
  - but they're being searched in /opt/products
- solution:
  - last thing in the install section, create a symlink
  - `/opt/products/%{name}/%{version} -> %{buildroot}/opt/products/%{name}/%{version}`
  - clean up afterwards...
- what if we're doing a DESTDIR install, and the previous release is installed ?
  - no problem, the files are there, dependencies will be ok
    - unless you rebuild because something was missing ...

Finally: Building the Packages

- you copied and adapted a SPEC file, input files are present
- you use the right find-requires script
  - `%define __find_requires /opt/products/lib/prpm/find-requires`
- now you can run **`prpm -ba --target target specfile`**
  - always use prpm (important for build requirements)
  - targets: DL4: i386, DL5: i586, (Solaris: sparc)
- output goes to **`/usr/src/packages`**:
  - **SRPMS/*name*-*version*-*release*.nosrc.rpm**
  - **RPMS/*target*/*name*-*version*-*release*.*target*.rpm**
  - **RPMS/*target*/*name*-default-*version*-*release*.*target*.rpm**
  - **...**

The Package Repository

- similar to the local RPM directory structure:
  - rooted at **`/afs/cell/packages`**
  - Zeuthen has a link **`/packages -> /afs/ifh.de/packages`**
  - **SOURCES (for tarballs not in the nosrc.rpm)**
    - **SOURCES/perl, SOURCES/root, ...**
  - **SRPMS (the nosrc.rpm)**
    - **SRPMS/perl, SRPMS/root, ...**
  - **RPMS/@sys (for platform specific .rpm)**
    - **RPMS/@sys/perl, RPMS/@sys/root, ...**
  - **RPMS/noarch (for platform specific .rpm)**
    - **RPMS/noarch/acroread, RPMS/noarch/pine, ...**

Repository: Product Subdirectories

- these are arbitrary (put pine into outlook/, if you like)
  - although it helps us humans if they make some sense
- our tools (ppm) use databases about all repository content
  - **RPMS/@sys/RPMDB**
    - an RPM database with all info about all packages
      - files, dependencies, conflicts, ...
      - for the platform and the platform independent ones
    - also contains a locatedb for finding the package files
      - this relies on correct file names of packages!
  - yes, dzpm would be perfect for this purpose
    - if HH would ever deploy it :-(

Committing a Release to the Repository

- note that's different from "copying files"
- a release consists of ALL files produced in a build
- a committed release is immutable
  - no changes to anything once it's in the repository
    - only permanent deletion is acceptable
      - still, name-version-release must not be reused
  - if you make a compatible change, that's a new release
  - if you make an incompatible change, that's a new version
    - or a different product
  - the spec file for name-version-release is identical across all platforms (use conditional macros)

Repository: Tools

- dry-run is the default for all tools
  - the switch `-x` makes it actually do something
  - all tools are located in **`/opt/products/sbin`**:
  - **`create_ppmdb`**
    - complete DB rebuild, slow - do not use
  - **`add_prpm`** ***`specfile`***
    - parses the SPEC, determines which RPMs were produced
    - checks for enough free space
    - copies all RPMs (including the (no)src.rpm, but NOT the sources)
      - destination defined in **`/afs/.ifh.de/packages/subdirs.def`**
    - releases all volumes involved

Repository: Tools

- if you committed a noarch package (platform independent)
  - **`add_prpm`** updated the database for your current platform
  - but not for the others
- **`update_ppmdb`**
  - updates the DB (much more efficient than create_ppmdb)
    - for your current platform
  - use on other platforms after noarch package committed
  - use after deleting a release (but think twice before)
  - use after restructuring the repository (new directories...)

Repository: Volumes

- AFS Volumes (& mount points) have to be created manually
  - afsadmin will help
- Volume names are arbitrary, the tools don't care
  - but stick to the current scheme
- Volumes for product subdirectories are optional
  - as long as parent does not grow too large
    - it's easy to change this later:
      - **`mv pine pine.BAK`** (now create volume and pine mountpoint)
      - **`mv pine.BAK/* pine; rmdir pine.BAK`**
      - **`arc vos release $PWD/pine; arc vos release $PWD`**
      - no need to update the DBs

Managing Installations

- now that we can
  - build releases
  - commit them to the repository
- we probably want to install them
  - on the reference installation (in AFS)
  - on the clients (fully locally, or mixed mode)
- could all be done manually, but that's tedious
- so we created ppm, the "products package manager"
  - the all-in-one(der)
  - everybody hates it - for good reasons ?

ppm Basics

- two basic modes of operation
  - query: `ppm -q`
  - change: `ppm -x file1 file2 ...`
    - default (without `-x`): dryrun mode
- by default, it works on client installations only
  - will refuse to run if `/opt/products -> /afs/.cell/...`
- the `-M` switch (think: master mode)
  - makes it work on reference installation
- the `-v` switch
  - makes it verbose (ppm is very silent without it)

ppm Documentation

- is better than commonly believed...
- `perldoc ppm`
  - everything you actually need to know
- `prpm -q --docfiles ppm`
  - points you to `/opt/products/ppm/0.9/doc/ppm.cf`
  - example configuration with lots of comments
- verbose dyruns
  - allow playing with it and understanding what it does

ppm: Queries

- `ppm -q`
  - lists all installed packages
- `ppm -q name`
  - lists all installed versions of package *name*
- `ppm -q /regex/`
  - lists all packages with a name matching regex
    - a perl regular expression
    - supports modifiers `i` (case independent) and `x` (extended)
      - `ppm -q /xml/i`
      - `ppm -q '/^ perl .+ xml/x`
      - `ppm -q '/^ perl .+ xml/ix`

ppm -q : Example Ouput

```
[arwen] ~ % ppm -q root
type   name  version       release  links
=========================================
link   root  3.02.07_gcc2  1
link   root  3.05.07       3        default
link   root  3.05.07_gcc2  2
link   root  3.10.02       3
link   root  4.00.00       1
local  root  4.00.00_ic8   1

[arwen] ~ % ppm -q '/^ perl .+ xml/x'
type  name                    version  release  links
=====================================================
link  perl_5.8.0-libxml-perl  0.07     1
link  perl_5.8.2-libxml-perl  0.07     1        default
```

ppm in Change Mode

- **ppm -vx *file1 file2 ...***
  - reads all config files given on command line
    - these completely define the desired final state
  - determines the current state (installed packages)
  - determines which packages are available (repository DB)
  - determines which packages are on the reference installation
  - checks sanity of desired final state, sanitizes if possible
    - resolve dependencies
    - change children to link if parent is link
    - change install type to local if not in reference installation, ...
  - does whatever needs to be done (unless in dryrun mode)

ppm Configuration Files

```
# these are all just comments

# (blank lines are ignored)
# EOF
```

- that's a perfectly valid input for **ppm**
- if it's the only input, **ppm** will happily delete each and every installed package
  - you defined the desired final state to be empty
  - you get what you ask for...

Another Valid Config File

```
# the next line starts a section
[install]
local     root   3.05.02    3   default
link    root   3.10.02    2
# EOF
```

- this one makes slightly more sense
  - but **ppm** will still delete itself, and perl, and ...
- sections switch the way input lines are interpreted
- in an **[install]**section, every line
  - adds or changes packages
  - possibly many packages

About Sections

- each section type may be present 0, 1, or more times
  - in one ore more input files
- duplicate and redundant input is ok
- conflicting input is ok
  - the last one wins
- allows
  - host-, group-, whatever-specific variation of base config
  - delegation
    - `ppm -x /etc/ppm/*.cf /etc/group/pkg.cf /etc/ppm/crit.cf`
    - last one can override catastrophic group config ...

Section Types

- **`[sysdef]`** - default installation flavour (local or link)
- **`[singlet]`** - packages that can be installed 0 or 1 times
- **`[forcelocal]`** - packages that can only be installed locally
- **`[forcejustdb]`** - packages only suitable for --justdb
- **`[mandatory]`** - define how to validate a complete selection
- **`[read]`** - include other config files
- **`[install]`** - add pckages to the selection
- **`[remove]`** - remove packages from the selection

[sysdef]

- defines the default installation type
  - final value is determined in first pass over all files
    - 1st input file (base configuration):
      - `[sysdef]`
      - `link`
      - `[install]`
      - `#type  name  version  release`
      - `Sysdef  root  3.05.07  3`
      - `...`
    - 2nd input file (say, host specific):
      - `[sysdef]`
      - `local`
  - combined result: local installation of all sysdef packages

[singlet]

```
[singlet]
glue
glue-devel
# or:
/^glue-/
```

- packages that cannot be installed in multiple versions
- either a verbatim package name
- or a perl regular expression enclosed in //
  - matches package name only

[forcelocal] & [forcejustdb]

```
[forcelocal]
pine
/^pam-/
[forcejustdb]
/^glue-/
```

- packages only suitable for a certain install type
- verbatim name or regex in // matching name
- complete list determined in first pass over all files
  - pine will be installed locally:

```
[install]
link pine 4.58 7
[forcelocal]
pine
```

[read]

```
[install]
...
[read]
/etc/local/group/ppm/*.cf
[read]
/etc/ppm/crit.cf    privileged
```

- include other files
- globs are guaranteed to be sorted alphabetically
- if token **privileged** is present, file may contain privileged sections (if file itself is privileged)
  - privileged sections: singlet, forcelocal/justdb, mandatory

[install]

```
[install]
#type   name        version   release   links
sysdef  maple       7         1         default
link    maple       8         2         default,alias

local   pine        4.58      7         default
link    pineconfig  1.0       2         local
```

- type: local, link, or sysdef
- name: verbatim, or a regular expression in //
- version/release: verbatim, or more sophisticated expression
- links: any combination of default, local, alias making sense
  - "" or "default" or "default,alias" or "local" or "local,alias"

[install]: details

```
[install]
#type   name             version   release    links
sysdef  /^perl_5.8.2-/   !         !          default

link    openssl          /./       !
local   openssl          <0.9.7,!  !          default
```

- all perl 5.8.2 modules, latest release (!) of latest version (!)
  - and their defaults
- all available openssl versions (/./), latest release (!)
  - as links
- latest (,!) version of openssl-0.9.6 (<0.9.7), latest release
  - locally, with default

Expression Evaluation

- left to right - think "filter"
  - name = `/^perl_5.8.2-/`
    - all packages with matching name, all versions, all releases
  - version/release = `!`
    - latest
  - version/release = `*` (new, experimental)
    - latest if local
    - as on reference installation, if link
  - version = `<0.9.7,!`
    - comma separated list evaluated left to right - think "filter"
- read the docs - or keep your config simple

Lines Matching Mupltiple Packages

- `sysdef /^perl/ /./ ! default`
- same effect as all corresponding lines for matching packages
  - sorted by package name (by alphabet)
  - then sorted by version number (by versioncmp)
  - then sorted by release
- default, local, alias are handled as attributes
  - default, local can only be set for a single version per name
  - the last one wins
  - => set on latest version & release of any matching package

[remove]

```
[install]
# all root versions, latest release, latest default
sysdef  root   /./            !   default

[remove]
# our group needs only one old version
root    /./    /./
[install]
local     root   3.02.07_gcc2  !   default
```

- remove packages from selection at this point
- no type, of course
- otherwise, works like **[install]**

[mandatory]

```
[mandatory]
#type    name            version   release   links
any      perl            >=5.8.0   /./       default
any      ppm             /./       /./       default
justdb   glue            /./       /./
local    openssl         <0.9.7    /./       any
```

- each line defines a sanity check of the final selection
  - final list determined in 1st pass over all input files
- test succeeds if any matching available package installed
  - with compatible type & default links
- here we make sure **ppm** works, and some local openssl 0.9.6x
- if any test fails, **ppm** will not change the installation

Wildcard Warning

- the wildcards are being used
- be careful what you dump into the repository
  - it may be installed on all clients soon
  - even if the configuration files aren't changed
- you are responsible for your packages AND the configuration
- if in doubt, define explicit release and version numbers
- if in doubt, use * instead of ! for release
  - prevents unwanted local installs
- start a dryrun on a typical client after each commit

ppm Master Mode

- the -M switch
  - makes ppm work even if **/opt/products** is a link pointing into the AFS
  - sets sysdef to local, no matter what the config says
  - will release all AFS volumes after work
    - mount points expected at top, name and version level
    - if root-3.10.02 was installed or updated, checks for mounts here:
      - /afs/.ifh.de/@sys/products/root/3.10.02
      - /afs/.ifh.de/@sys/products/root
      - /afs/.ifh.de/@sys/products
    - volumes are optional

ppm: Loose Ends

- current version 0.9-16 works reliably
- still lacks some features
  - logging
  - email notification (notify/warn/alert)
  - repairing messed up RPM DBs
    - sometimes, name-version-release "installed" twice
      - stalls some actions
    - version 0.9.17 can detect and correct this state
  - the semantics of the * wildcard is not finalized
    - use for release number is ok though

ppm Usage in Zeuthen

- sue feature products
  - can bootstrap a client installation (sue relies on perl though!)
  - populates **/etc/ppm** with .cf files from sue archive
    - **$OS_ARCH/etc/ppm**
    - **$HOST/etc/ppm**
  - runs **ppm** if any input is more recent than a lastrun file
    - new state of /etc/ppm
    - new available packages DB
    - new reference installation DB
  - cheap if nothing to do
    - runs in sue.hourly

Conclusions

- one year after the kickoff meeting
- the **`/opt/products`** scheme was
  - refined
  - implemented
  - deployed
- in operation on DL4 & DL5, Solaris (2.6, 8) proof of concept
- we built hundreds of packages, being used successfully
- the effort is starting to pay off
  - cheap, fast, reliable build & rollout of new versions/releases
  - "can I have that on my notebook ?" - the answer now is "yes"