KTL's Suggested Implementation of Software Environment Use Cases

I'll take Ray's use cases and describe how I'd like to see them handled. At the end I list a summary of the concepts and commands implied by this implementation of the use cases. See also Packaging System and Russell Owen's svn-related Setup Use Cases.

Production Execution

1. Setup for Application Execution

Ray does not go into the details of how these packages are installed (although there is related information in use case 4).

For typical application execution, the installed packages should be tagged versions retrieved from the repository. In my ideal world, such versions are retrieved as binary archives for the user's platform where possible (for efficiency) and built from source otherwise. If they are built from source, the known-good snapshot of the dependent packages' versions must be used while building, not anything dependent on the installing user's configuration. The command used for accessing the repository should be "`eups fetch`".

All such installed packages should be identical for a given platform, no matter which user performs the installation. Accordingly, there seems to be no reason to have per-user installation directories for this case. Note that this does not mean that all users should have the same set of "current" (or default/preferred) versions; those may vary on a per-user or even per-shell basis.

Building a package from source must check that dependent packages are activated, and, ideally, that sufficient files are available from those packages to complete the build. This can be accomplished by having each package specify its "interface", which consists of zero or more header files, zero or more Python packages, and zero or more shared libraries.

Activating a package of course needs to automatically activate compatible versions of its dependent packages, starting with the current/preferred version but selecting another if required. Activating can automatically retrieve packages if they have not been installed to further simplify the end-user's task.

> **Proposed DC3 command:** `activate dc3pipe --stable`

2. Setup for executing a particular version of an application

If the installed package is under rapid development, in the current system it may have been installed from a checked-in svn working copy rather than retrieved from the package repository, hence the "1.2+svn4455" designation. There are two problems with this: automatically generating the version number is difficult, and ensuring a reproducible installation is also difficult, because the "`setup -r .`" command typically used for defining the build configuration uses whichever packages are labeled "current". In DC2, this was not an issue because all developers working in this style used the same physical installation of the package, but this is not workable during normal distributed development. Besides solving the above two problems, a mechanism is needed for developers on other machines to get access to the same version.

My preferred solution is to allow `eups fetch` for svn versions. I would disallow "`scons install`" and replace it with "`eups install`", making `eups` the sole command used for maintenance of the installation tree. "`eups install`" of a checked-in svn working copy would install a version automatically numbered using a "base tag" version number. This command would not be allowed if files existed that had not been checked in.

Importantly, an additional command, "`eups publish`" would also upload a lightweight package description to the package repository. This description would consist of a snapshot of the dependent package versions activated for the build and the svn revision of the trunk in use. This should be sufficient information for "`eups fetch`" to re-create the developer's build of the package, producing the same situation that the original developer produced with the "`eups install`" command. The package description is also sufficient information for an automated build system to create a binary version of the package for download by "`eups fetch`" for efficiency. Finally, with a tag version instead of an svn revision, this description could even be used for making tagged versions available to the package repository.

Note that "`eups install`" must rebuild the package if any dependent package activations have changed since the last time the package was last built with `scons`; it may wish to always rebuild the package from scratch.

The "base tag" version number allows us to have revisions from release maintenance branches or even directly from ticket branches. Such svn versions will always correspond with their preceding tagged versions, allowing humans or software to easily determine the suitability of such versions as dependencies. The "base tag" for a given version could be specified in the table file or another location, such as an svn property. Right now, a similar version number is in the `.pacman.m4` file, but I believe this is the version of the *next* release, not the version of the *last* release, making this much more difficult to maintain. Having this base tag version would allow versions of the form "`{base tag}+svn{revision number}`" to be auto-generated, simplifying comparisons and improving human understandability. The base tag version could be updated automatically by a tool whenever a tag copy is made.

Once again, there is no need for per-user installation directories in this case, as all installations are identical and installing a version cannot negatively impact another user's configuration.

> **Proposed DC3 command:** `activate dc3pipe 3.4`

3. Setup for executing an application using a particular version of a dependent package

The proposal here is to add a `--keep` option to `setup`/`activate` in order to allow prior specifically-chosen versions to be kept, instead of having them replaced. Alternatively, the existing method of overriding versions in dependency order can still be used. In either case, activating a specific version of package X must check that that version satisfies the requirements of all currently-activated packages that depend on package X.

> **Proposed DC3 command:** `activate daf/persistence 3.12; activate dc3pipe --keep`

Installation

4. Installing a tagged release into the stack

`setup -r .` does not ensure a reproducible build environment, as described above in use case 2. My preference here is again to mandate the use of `eups fetch` for installing tagged (or untagged) versions.

> **Proposed DC3 command:** `eups fetch daf/persistence` (for stable version)

5. Installing a beta version of a package

See use case 2, including the caveat about branch versions. The "beta version" would typically be marked as preferred for the user, so that manual version specification would not be necessary for every shell/login, but this marking should not be shared by other users on the same machine. It may be desirable for a developer to mark a particular "beta version" as the preferred one for all other developers, but not for application users.

> **Proposed DC3 command:** `eups fetch daf/persistence --beta`
>
> **Proposed DC3 commands for developers:** `cd daf/persistence; eups install; eups publish daf/persistence --current; eups mark daf/persistence --current --beta`

6. Reinstalling a version of a package

I don't think it should ever be necessary to replace an installed version. This is dangerous in many respects: it may break running programs, it may lose history, it may break provenance.

> **Proposed DC3 command:** None

7. Removing a package

Removing obsolete packages should be supported. "`eups install --remove`" seems like a reasonable way to do this.

> **Proposed DC3 command:** `eups install --remove daf/persistence 3.12+svn6235`

Development

8. Adding and testing new features into a package

The main thing here is the need to activate the current working directory as the source for files for the package being built and remember that this directory should be the source for this user ("install devel" in Russell's terminology). This requires activating a package directory outside the installation tree that may not ever have been built. Note that the directory needs to be activated, not the files in it, because those files may not exist and the set of files may even change during development. (This makes activation of an devel-installed package more complicated for a link-forest implementation, for example.) The devel-installed package version will typically also be marked as the preferred version for the developer user, but it should not be marked as preferred, or even visible at all, to any other user. The devel-installed package must have at a minimum a table file specifying its dependencies (and perhaps its "base tag" version); it may also have an interface defined. Activating the devel-installed package, like activating any package, should automatically

activate the appropriate dependent packages.

Since we are calling this a form of installation, we should use `eups` to perform this action for consistency. "`eups install --devel`" might be acceptable. A devel-installed package need not have all of its files checked in to be used.

It may be necessary to have more than one devel-installed version of a given package for a given user if the developer is working on multiple tickets, perhaps with different base tags, at the same time. Only one would be preferred, but each could be activated in its own shell. Thus the devel-install command must take a user-specified name to distinguish between these versions.

> **Proposed DC3 command:** `eups install --devel`

9. Adding and testing new features into several packages simultaneously

Multiple devel-installed packages may need to be activated and marked as preferred at the same time.

> **Proposed DC3 commands:** `eups install --devel; cd ../../daf/base; eups install --devel`

Concepts

The above use cases lead to the following concepts:

Installation status:

- Tagged package versions installed on the machine.
- Svn revision package versions installed on the machine.
- Tagged or svn revision package versions published to the project-wide package repository.
- Devel-installed, per-user package versions.
- No version can be activated without being installed or devel-installed.

Preferred status:

- Versions preferred for application users: "stable".
- Versions preferred for developers: "beta".
- Versions preferred by user: "current".
- Versions activated by user in this shell: "setup"/"activated".

Package contents:

- Dependencies: table file with ranges.
- "Base tag" revision: table file.
- Interface: interface file.
- Repository description: dependency snapshot plus tag version or svn revision and svn URL.

Proposed Command Syntax

Inquiry Commands

`eups list [PACKAGE [{VERSION, --current, --active}]]`

Lists packages installed on the machine with versions. If a package is specified, lists only versions of that package. If a version or option is specified, lists only that version of the package (if any). A glob (like for shell pathnames) may be used instead of a package name to list all packages matching the glob.

`eups list {--current, --active}`

Lists all current or activated package versions on the machine.

`eups list {-r, --repository} [PACKAGE [{VERSION, --stable, --beta, --all}]]`

Lists all available packages from the project-wide package repository with available versions. Lists tagged versions only unless `--devel` or `--all` are specified.

Installation Commands

`eups fetch [PACKAGE [{VERSION, --beta}]] [--nocurrent]`

Retrieves a package from the project-wide package repository, building it from source if necessary, and installing it into the machine's standard installation tree. The version defaults to the one marked stable, if present, then the one marked beta, then the latest tagged version, and finally the latest version if none of the previous ones exist. Automatically marks the fetched version as current unless `--nocurrent` is specified.

Building from source uses the exact versions of the dependent packages from the installation snapshot.

`lsst-tag VERSION [DIRECTORY]`

Tags a working copy of a package as a release version. Checks to make sure that all files are checked in. Modifies the "base tag" (whether in a file or a property) and checks that in. Makes the tag copy in the svn repository. Defaults to the current directory.

This command would be used by the package owner as part of a release process; it performs the necessary svn functions for release. Before tagging, the release process would build and test the package. After tagging, the process would install and publish the package and possibly mark it as stable (see below).

`eups install [DIRECTORY] [--nocurrent]`

Installs a package working copy as a tagged or svn version into the machine's standard installation tree. Defaults to the current directory.

First, ensures that all files are checked in and from the same svn branch. Rebuilds the package if the activated version of any dependent package has changed since the last build. Takes a snapshot of the activated versions of all dependent packages. Also records the svn URL of the working copy.

If the directory is an unchanged tag directory, installs with version `BASETAG`. If in another type of working copy, installs with version `BASETAG+svnREVISION`. Automatically marks the installed version as current unless `--nocurrent` is specified.

`eups install {-d, --devel} [NAME] [--nocurrent]`

Devel-installs the current directory for this user. The package name is auto-detected. This devel-install replaces any other directory devel-installed for this package with the same name. The version is `BASETAG+svnd`, or `BASETAG+svnd+NAME` if a name is specified. This command does not build the package. No snapshot of dependent package versions is made. The svn URL of the working copy is not recorded. Automatically marks the devel-installed version as current unless `--nocurrent` is specified.

`eups install --remove PACKAGE VERSION`

Removes a version installed on the machine (rarely used). For devel-installed versions, removes the knowledge of the installation but does not touch any files in the package directory.

`eups publish PACKAGE {VERSION, --current}`

Makes a tagged or svn version installed on the machine available to the project-wide package repository. Devel-installed versions cannot be published. Requires a specific version or `--current` to use the version marked current. Copies the package description, including the dependent package snapshot, to the package repository.

Note: There could be a potential conflict if two developers install versions of a package on their own machines, but using different versions of dependent packages, and then both try to publish their versions to the project-wide package repository. Presumably either one could be used, but this is still a case where different developers can have different ideas of what a single package version means.

`eups publish --remove PACKAGE VERSION`

Removes a published version from the project-wide package repository (rarely used).

`eups mark {stable, beta} PACKAGE {VERSION, --current}`

Marks a published version in the project-wide package repository as "beta" or, if the version is a tagged version, allows it to be marked "stable". Any version previously marked with the same mark has its mark removed. Requires a specific version or `--current` to use the version marked current.

`eups mark {stable, beta} --remove PACKAGE VERSION`

Removes a mark (rarely used).

`eups current PACKAGE VERSION`

Marks a locally installed version as "current" for this user. Given the automatic marking as current for `fetch`, `install`, and `install --devel`, this command is not expected to be used frequently. It is distinct from `eups mark` since it operates on a per-user basis, rather than on the project-wide package repository.

Activation Commands

`activate PACKAGE [VERSION] [{--stable, --beta}] [{--keep, --exact}]`

Activates a package, checking all dependencies. If no version is given, uses the current version if possible; otherwise uses the latest version that works. If a specific version is given or `--stable` or `--beta` is specified, tries to use that version if possible; fails if not. If a specific version is given *and* `--stable` or `--beta` is specified, tries to use the specific version while using `--stable` or `--beta` for dependencies (see next paragraph).

This command also activates dependent packages if needed. If `--exact` is specified, the exact versions used when the package was installed are used. Tries to use the already-activated version of dependent packages if `--keep` is specified; fails if this is not possible. Otherwise, if `--stable` or `--beta` is specified, tries to use those versions of dependent packages first. Next tries the current version. Finally, uses the latest version that works if none of the previous versions are possible and fails if no versions work.

Calls `eups fetch` if any package is not installed to retrieve the package from the project-wide package repository.

`deactivate PACKAGE`

Deactivates activated version of package and any packages depending on given package.

Comments

Comment by rowen on Tue 13 May 2008 07:13:33 PM CDT

I like this proposal. I just want to point out a few things:

- There is no ability to install a package in such a way that the files get copied but only one user can see them. I like the simplification.
- There is no ability to change the mark (e.g. "beta") of a machine-wide installed package. I think we can live with this but it makes me a bit nervous. We can always add it back in if we find we need it.
- If this requires too much interconnection between eups, scons and svn we may want to add back a shell like "lsstpkg" to execute some of these commands.

Comment by rowen on Wed 14 May 2008 11:22:44 AM CDT

One more quibble: I would prefer "activate" not call "package fetch" if a package is missing. A suitable error message would suffice. Though if you want to make it that smart then please at least ask before connecting to the network. Issues:

- The user may not have permission to install packages.
- The user may have made a mistake and tried to set up the wrong thing.
- It should be very rare, right? Surely it can only occur if somebody actively messes up a package repository by removing needed packages. If so, I see little value in adding much smarts to handle it.

Reply by ktl

The main reason for `activate` calling `fetch` was to make preparation for running a pipeline on a bare machine (with eups) one command. It's not a big deal to make it two (`fetch` and then `activate`). I can see it being used if someone is not keeping up to date with some packages and then needs them one day.

If all packages are identical (especially if they are binary tarballs), permission to install packages should be granted widely. Installing a package due to incorrect activation would also be fast if packages are binaries; it would be a pain if rebuilding from source.

Comment by rhl on Tue 03 Jun 2008 05:44:58 PM CDT

See wiki:RHLSetupComments --- too many to put in here, as most are interspersed with K-T's text.

Add comment

Your email or username: