[Software Development]

Package Management

Davide Balzarotti

Eurecom – Sophia Antipolis, France

Software Development Tools

1. Configuring and Building the program
   - ✓ GCC
   - ✓ Makefiles
   - ✓ Autotools
2. Packaging and Distributing the Application
3. Managing code
4. Debugging and Profiling

Software Packages

- Packages are bundles of <u>software</u> and <u>metadata</u>
    - Metadata specify:
        - Package info (such as the software's full name, description of its purpose, version number, vendor)
        - List of dependencies necessary for the software to run properly
    - Software can be in both source or binary forms

- A package management system is a collection of tools to automate the process of *installing, upgrading, configuring*, and *removing* software packages from a computer
- Package managers usually provide the users the ability to install and upgrade software over the network in a seamlessly integrated fashion

Package Management

"Package management is the single biggest advancement Linux has brought to the industry"
- Ian Murdock

- A package management system is *NOT* a software installer
  - It is not distributed with the application, but it is typically part of the operating system
  - It uses a single installation database and manage packages in a uniform way (installers instead come in different flavors)
  - It can verify and manage all packages on the system
  - It can automatically find and fetch packages from known software repositories

Why Packages

- Manage dependencies
  - What prerequisites do the packages have?
  - Does it substitute another package?
- Manage conflicts
  - What if two programs install/modify the same file?
- Manage upgrades
  - What if user has customized configuration?
  - What if file ownerships/permissions have changed?
  - What if user needs the old and the new versions at the same time?

Each Distribution has its Own

- There is no standard package manager in Linux :(
- Different distribution adopt different systems
  - Tarball-based (`.tgz`)
  - RPM (RedHat Package Manager) (`.rpm`)
  - DEB (Debian Package Manager) (`.deb`)
  - Solaris packages (`.pkg`)
  - BSD ports
  - Gentoo Portage
  - ...

In the beginning there were Tarballs

- Tarball files are the old-fashioned way of distributing software in Linux/Unix
  - Compatible with all distribution
  - Still the preferred way to distribute source codes
- Main package format used by Slackware and Gentoo
  - Slackware uses tarballs containing the software sources and some extra file (`slack-desc` and `doinst.sh`)
    - No support for tracking or managing dependencies
    - Rely on the user to ensure that the system has all the supporting system libraries and programs required by the new package
  - Gentoo has a more sophisticated package manager (more later)

RPM

- RPM (RedHat Package Manager) (`.rpm`)
  - Originally introduced by RedHat
  - Now adopted by many other distributions (Fedora, Mandrake, SuSe...)
  - RPM is the baseline package format of the Linux Standard Base
- Normally used to distribute pre-packaged binary software
- It can keep track of package dependencies
- Managing the packages:
  - Individual package can be managed by the rpm command line tool
  - Automated updater are available to automatically retrieve packages and computes dependencies to figures out what things should occur in order to safely install, remove, and update rpm packages
    - Yum (Yellowdog Updater, Modified)

Debian Packages

- DEB (Debian Package Manager) (`.deb`)
  - Introduced by the Debian distribution
    (and now used by any debian-like distros like Ubuntu)
  - Debian was the first linux distribution to rely on package interdependencies to perform reliable system upgrades
    - It's now the second oldest distribution still alive
      (slackware is leading by few weeks)
- Managing the packages:
  - Individual packages can be managed by the `dpkg` command line tool
  - Automated updater are available
    - APT (Advanced Package Tool) – command line
    - dselect (ncurses front-ends to dpkg)
    - Update-manager (graphical application for ubuntu)

BSD Packages and Ports

- A port for an application is a collection of files designed to automate the process of compiling an application from source code
  - Download the port, unpack it and type make in the port directory
  - Each port's Makefile automatically fetches the application source code, (either from a local disk, CD-ROM or via ftp), unpacks it on your system, applies the patches, configures it, and finally compiles it.
  - Gives the user a lot of flexibility
- Packages are pre-compiled copies of a software
  - Can save the user a lot of time when installing large applications
  - `pkg_add, pkg_delete, pkg_info` are used to manipulate packages
- Both packages and ports understand dependencies and are able to automatically install the required software

Gentoo's Portage

- Modeled on the ports-based BSD distributions
- The Portage tree is a collection of ebuilds
- ebuilds are scripts containing the instructions to download, patch, compile, and install packages
  - Dependency checking, extreme customization
  - Original source tarballs are downloaded
  - The user specifies what he wants, and the system is built to his specifications
- The emerge tool is used to calculate and manage dependencies, execute ebuilds and maintain the local Portage tree and database of installed package
  - ebuilds are executed in a sandbox environment to protect the system from software executed by the ebuild. The resulting binaries are only merged after a successful build and sandboxed install

Short Guide on Managing rpm and deb

- Install a package

```
> dpkg -i file.deb
> rpm -i file.rpm
```

- Update a package

```
> dpkg -i file.deb
> rpm -U file.rpm
```

- Remove a package

```
> dpkg -r package_name
```

  - remove (-r) does not delete the configuration file (for further re-installation)

```
> dpkg -P package_name
```

  - purge (-P) removes everything

```
> rpm -e [--nodeps] package_name
```

  - --nodepts forces the system to remove a package even if other software depends on it

Querying

```
> dpkg -l
```
or
```
> rpm -qa
```

List all the packages installed in the system

```
> dpkg -L pkg_name
```
or
```
> rpm -q -l pkg_name
```

List all the files installed by a particular package

```
> dpkg -S filename
```
or
```
> rpm -qf filename
```

Show the package that installed a certain file in the system

```
> dpkg -p pkg_name
```
or
```
> rpm -qi pkg_name
```

Print the details of a particular package

Using APT

- APT is a set of tools (apt-get and apt-cache are the most important) to retrieve, install, and manage relations (especially dependencies) between packages
- APT does topological sorting of the list of packages to be installed or removed and calls dpkg in the best possible sequence.
- Basic functionalities:

```
> apt-cache search keywords
> apt-cache show package_name
> apt-get install package_name
> apt-get [--purge] remove package_name
```

Upgrading The Entire System

`apt-get update`

Resynchronize the local package index files from their sources. The indices of available packages are fetched from the locations specified in `/etc/apt/sources.list`

`apt-get upgrade`

Update all the installed packages to the newest version available. Under no circumstances are currently installed packages removed or new packages installed in the system

`apt-get dist-upgrade`

Upgrade all packages and in addition smartly decide which new packages must be installed and which one must be removed

Using YUM

- Inspired by APT, provides similar functionalities to work with rpm packages
- Yum automatically synchronizes the remote metadata to the local client
- Use:

```
> yum list [available | installed | updates]
> yum install package_name
> yum update package_name
> yum remove package_name
```

  - It removes package_name and all packages in the dependency tree that depend on package_name, possibly irreversibly as far as configuration data is concerned

```
> yum info package_name
> yum search keywords
```

A Quick Look inside a rpm Package

- rpm are binary files that require special tools to be inspected
- Extract package dependencies:

```
rpm -qpR file.rpm
```

- Extract package information:

```
rpm -qip file.rpm
```

- Extract package install and uninstall scripts:

```
rpm -qp --scripts file.rpm
```

- Extract package files:

```
rpm2cpio package.rpm > package.cpio
cpio -idmv < package.cpio
```

A Quick Look inside a deb Package

- `.deb` files are pure `ar` archives
- The content of a deb file can be extracted by:

```
ar vx file.deb
```

- The archive contains three files:
  - debian-binary → version of the deb file format (2.0)
  - control.tar.gz → contains special files like package description, file MD5s, pre- and post- installation scripts..
  - data.tar.gz → all the files to be installed with their destination paths

Checkinstall

- Software manually downloaded and compiled from sources is normally invisible to the package management system
  - Difficult to keep track of it
  - Hard to know which files are installed where
  - Difficult to remove
- CheckInstall keeps track of all files installed by a "make install" (or equivalent), creates a Slackware, RPM, or Debian package with those files, and adds it to the installed packages database
  - CheckInstall is not designed to produce packages suitable for distribution

Checkinstall

- Use:

```
> configure
> make
> sudo checkinstall
```

- It is possible to include the package documents that will be installed in `/usr/doc/<package_name>`

```
> mkdir doc-pak
> cp README INSTALL COPYING Changelog doc-pak/
> sudo checkinstall
```

- A copy of the package is left on the current directory
- The package can easily be removed by using the system package management system
  - For instance, under debian

```
> sudo dpkg -r packagename
```