Advanced Usage of the Debian Packaging System

Richard Darst
With contributions from #debian-nyc

September 15, 2010

Prepared for
New York Linux User's Group

Contents

Goals of this talk

- Show you what tools are available
- There is **not** enough time to discuss these tools in detail
- This is **not** designed to be a tutorial
- Once you know what exists, you can read up on it yourself easily, and ask us for help.
  - `man` *programname*
  - `/usr/share/doc/`*packagename*`/`
  - Things in this presentation which are uncited will usually come up first on an Internet search.

What is a package?

Packages are the fundamental unit of a Debian installation

- Every file on a system belongs to a package
- ... but some are automatically generated or exceptions
- Packages include files, but also
  - Control information (descriptions, dependencies, versions)
  - Installation/removal/**upgrade** scripts

The Debian Policy

The Debian Policy is rules which packages (and the system in general) must follow.

- Debian's quality-control document
- Based on years of experience
- Formed by convention and mailing list discussion
- `http://www.debian.org/doc/debian-policy/`

Examples of things in policy

- Build interface for packages
- Definitions of dependency levels
- User interaction (GUI menu interface, defaults, ...)
- Location of files (POSIX File Hierarchy Standard)
- Installation/removal/upgrade scripts

Policy leads to quality

Policy is considered to be one of the strengths of Debian:

- Makes things uniform, so we can...
- Port to many architectures
- Detect bugs by archive-wide automatic rebuilds
- Consistent user interface
- Have package upgrades that work

Suites

A **suite** is a section of the Debian archive of packages that go together.

- Suites for releases:
  - Stable - Unchanging released version
  - Testing - More stable than unstable, becomes stable upon a release
  - Unstable - New packages go here, use at your own risk
- These suites contain the same packages, but different versions
- Specialized suites: security, experimental, volatile, backports

Dependencies

Dependencies indicate the relationship between two packages.

- Depends, Pre-depends, Recommends, Suggests, Enhances, Breaks, Conflicts, Provides, Replaces
- Names are intuitive here
- Usually, "Recommends" of a package are automatically installed, while "Suggests" are not
  - Depends on package manager

Source vs Binary packages

Debian packages come in two main varities: binary and source

- As a user, you always deal with **binary** packages
- Binary packages are **.deb** files: compiled code.
- Source packages are **.dsc** file, plus `.orig.tar.gz`, `.diff.gz`, and more: Source code.
- Source packages are not installed directly
- `.udeb`: installer, `.tdeb`: translation

dpkg

dpkg is the low-level package manager. Things which it handles include:

- Ensures dependencies are met
- Installs files from packages
- Runs scripts for installation/removal/upgrade
- Maintains database of installed packages

dpkg operates on .deb files. Usually, you won't download and install .debs yourself.

APT

APT is the high-level package management system

- Maintains list of all available packages from the archive
- Resolves dependencies
- Downloads and hands install off to `dpkg`
- Different frontends: `aptitude`, `synaptic`
- APT replacements: `cupt`

APT is what you use for most things you'll install distributed by Debian.

debconf

Debian has a configuration management system for basic and medium customization.

- debconf (as opposed to DebConf, the conference) is the configuration management system
- Manages the configuration dialogs you see upon installation/upgrades.
- Maintains database for configuration between upgrades

$ **debconf-get-selections**

```
# Upgrading from GRUB Legacy.
grub     grub/migrate\_from\_legacy          note
# Time zone:
# Choices: Adelaide, Brisbane, Broken Hill, Canberra, Currie, Darw
tzdata  tzdata/Zones/Australia  select
# Do you want system-wide readable home directories?
adduser adduser/homedir-permission       boolean true
# What do you want to do about modified configuration file ?
# Choices: install the package maintainer's version, keep the loca
ucf      ucf/changeprompt          select  keep\_current
# Keymap:
# Choices: Programmer, latin1, latin1 - no dead keys
console-data     console-data/keymap/qwertz/german/standard/keymap
```

Version numbers

Version numbers are important since they indicate upgrades and preferred versions. Simple case:

$$\underbrace{10.16.2}_{\text{upstream version}} \text{ - } \underbrace{1}_{\text{debian revision}}$$

Epoch fixes problematic upstream version changes:

$$\underbrace{1}_{\text{epoch}} : \underbrace{10.16.2}_{\text{upstream version}} \text{ - } \underbrace{1}_{\text{debian revision}}$$

Extra modifier for stable updates, backports, other situations:

$$\underbrace{2}_{\text{epoch}} : \underbrace{3.2.5}_{\text{upstream version}} \text{ -} \underbrace{4\underbrace{\text{lenny11}}_{\text{extra modifier}}}_{\text{debian revision}}$$

dpkg based tools

For working and learning about installed packages or `.deb` files, use `dpkg` and its related tools.

- `dpkg` installs files and runs scripts
- Package naming convention:

$$\underbrace{\text{python-numpy}}_{\text{package name}}\_\underbrace{\text{1.3.0-4}}_{\text{vers. num.}}\_\underbrace{\text{amd64}}_{\text{architecture}}\text{.deb}$$

- `dpkg` state directories: /var/lib/dpkg/
  - /var/lib/dpkg/status
  - /var/lib/dpkg/info/*packagename*.{postinst,prerm,md5sums,conffiles,...
  - /var/lib/dpkg/available
  - /var/lib/dpkg/

dpkg state

dpkg maintains a database of state of installed and partly installed packages.

- A package can be: **not-installed**, **installed**, **config-files**, **unpacked**, **half-installed**, and some more.
- `dpkg --remove` → leave config files
- `dpkg --purge` → remove config files, too
- The other states are various combinations of installation scripts partially run
- `man dpkg`

Examining .deb files

If you have downloaded a .deb yourself, you can learn some about it before installing.

- To extract a .deb: dpkg-deb -x
- dpkg (really dpkg-deb) can look inside .deb files and tell you about them.
- List files with in a package:

$ **dpkg -c** *python-numpy_1.3.0-4_amd64.deb*

```
drwxr-xr-x root/root          0 2009-11-19 02:19 ./
drwxr-xr-x root/root          0 2009-11-19 02:19 ./usr/
drwxr-xr-x root/root          0 2009-11-19 02:19 ./usr/bin/
-rwxr-xr-x root/root        675 2009-11-19 02:19 ./usr/bin/f2py2.4
-rwxr-xr-x root/root        675 2009-11-19 02:19 ./usr/bin/f2py2.5
drwxr-xr-x root/root          0 2009-11-19 02:19 ./usr/share/
drwxr-xr-x root/root          0 2009-11-19 02:19 ./usr/share/man/
...
```

Examining .deb files

- List control information:

$ **dpkg** -I *python-numpy_1.3.0-4_amd64.deb*

```
new debian package, version 2.0.
size 1818486 bytes: control archive= 11769 bytes.
   1274 bytes,    22 lines      control
  30246 bytes,   363 lines      md5sums
    349 bytes,    11 lines   *  postinst             #!/bin/sh
    280 bytes,    10 lines   *  preinst              #!/bin/sh
   1219 bytes,    47 lines   *  prerm                #!/bin/sh
Package: python-numpy
Version: 1:1.3.0-4
Architecture: amd64
Maintainer: Debian Python Modules Team
<python-modules-team@lists.alioth.debian.org>
Installed-Size: 7640
...
```

Querying installed packages

Once a package is installed, a different set of commands is necessary to examine them.

- `dpkg -s` shows control information for a package
- `dpkg -l` lists packages and states

```
$ dpkg -l 'bash*'
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Cfg-files/Unpacked/Failed-cfg/Half-inst/trig-aWa
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: u
||/ Name           Version        Description
+++-==============-==============-================================
ii  bash           3.2-4          The GNU Bourne Again SHell
ii  bash-completio 20080705       programmable completion for the
un  bash-doc       <none>         (no description available)
```

List files within a package

You can list the files in an installed package with `dpkg -l`

- I often use this to get a hint about *how* to use a package after I install it
- Will not list files that were automatically generated from the package's scripts

$ **dpkg -L bash**

```
/.
/bin
/bin/bash
/etc
/etc/skel
/etc/skel/.bashrc
/etc/skel/.profile
/etc/skel/.bash_logout
/etc/bash.bashrc
/usr
/usr/share
/usr/share/doc
/usr/share/doc/bash
/usr/share/doc/bash/CHANGES.gz
```

Backup package selections

`dpkg --get-selections` lists installed packages.

- Back up package installation states
- Could be used to transfer selections to another computer
- This does not backup the "automatic/manual" database.

$ **dpkg –get-selections**

```
a2ps                                            install
acl                                             install
acpi                                            install
acpi-support-base                               install
acpid                                           install
acx100-source                                   install
adduser                                         install
alsa-base                                       install
```

- **Use with** `dpkg --set-selections`

Which package owns a file?

If you are looking at a file and want to know what package put it there, use `dpkg -S`

- These commands only search *currently installed* packages
- Next section gives a better way to search for files
- Will miss automatically generated files

```
$ dpkg -S /etc/bash_completion
bash-completion: /etc/bash_completion
```

```
$ dpkg -S /etc/debian_version
base-files: /etc/debian_version
```

Other places to look for information on installed packages

After installing a package, a common question is "How do I use it?"

- It is usually worth it to take a bit of time to learn the Debian way of using a package, instead of rolling your own configuration.
- List files in it: `dpkg -l` *packagename*
- Per-package documentation: `/usr/share/doc/`*packagename*`/`
  - `/usr/share/doc/`*packagename*`/NEWS.Debian.gz` - Major changes you should be aware of
  - `/usr/share/doc/`*packagename*`/changelog.gz` - upstream changelog
  - `/usr/share/doc/`*packagename*`/changelog.Debian.gz` - Debian changelog
- `/etc/defaults/`*packagename* - Sometimes useful for daemon-like packages

APT: Advanced Packaging Tool

APT is the companion to dpkg

- APT is a higher level than dpkg
- APT handles dependencies and downloading
- Configure mirrors and suites in /etc/apt/sources.list
- Stores it's state in /var/cache/apt/
- dpkg deals with what *is* installed, APT deals with what *can* be installed

APT is a set of libraries with different frontends.

Frontends to APT

There are different frontends to `APT` , and they can have slightly behaviors:

- `apt-cache`, `apt-get`
- `aptitude`
  - Smarter at resolving dependencies
  - Installs recommends by default (`aptitude --without-recommends` to change)
  - The currently recommended distribution upgrade program
- `synaptic`, KPackage, `apt-rpm`, and many more...

APT cache utilities

The first thing APT can do is search available packages

- Run apt-get update to refresh the list of what is available.
- Use apt-cache to search for packages and show packages

$ apt-cache search calculator

```
abakus - calculator for KDE
allegro-examples - example programs and demo tools for the Allegro
apcalc-common - Arbitrary precision calculator (common files)
apcalc-dev - Library for arbitrary precision arithmetic
...
```

$ apt-cache show abakus

```
Package: abakus
Priority: optional
Section: kde
Installed-Size: 728
...
```

APT dependency handling

APT handles dependencies intelligently (downloading them), while dpkg just fails if dependencies are not met.

- To find related programs, look at dependencies and reverse dependencies - you may find something that will help you
- I always just do this from the aptitude user interface

$ apt-cache depends units

```
units
  Depends: libc6
  Depends: libncurses5
  Depends: libreadline6
 |Depends: dpkg
  Depends: install-info
```

$ apt-cache rdepends units

```
units
Reverse Depends:
  octave-miscellaneous
  ibid
```

apt-file

You can do archive-wide searches of files within packages.

- dpkg -S only finds currently installed files
- apt-file searches all the archive (using Contents.gz) for files

$ **aptitude install apt-file**

$ **apt-file update**

$ **apt-file search bin/host**

```
apt-file search bin/host
bind9-host: /usr/bin/host
coreutils: /usr/bin/hostid
host: /usr/bin/host
hostap-utils: /usr/sbin/hostap_crypt_conf
hostap-utils: /usr/sbin/hostap_diag
...
```

Other APT package states

APT has more states that dpkg . dpkg just looks at installed or not, but APT has more

- Some APT frontends (aptitude was first, apt-get has it now, too) maintain a database of **manually installed** vs **automatically installed** as dependencies.
  - If you uninstall a package, its dependencies are automatically uninstalled, too.
  - This is why I used to recommend always using aptitude. Use "m"/"M" to adjust auto-flag.
  - aptitude vs apt-get databases here
- **Hold** - APT will never upgrade this package to a newer version.
- Recommends vs suggests
  - Recommends are usually automatically installed, suggests are usually not.

Pinning

Pinning indicates package preferences:

- If you mixed suites (Stable and Testing, backports), **pinning** can tell APT to prefer certain
  - suites
  - repositories
  - versions
  - and more
- `man apt_preferences`
- Example (Use testing and unstable, but prefer testing):

```
Package: *
Pin: release a=testing
Pin-Priority: 900

Package: *
Pin: release a=unstable
Pin-Priority: 800
```

Diversions and overrides

You can tell dpkg to preserve certain custom modifications.

- You want to change a file in a package, but have it **not** get overridden when you upgrade the package the next time
- Diversion: Tell dpkg to install a file in a different location, so that your can replace it and not be squashed on upgrades
  - `dpkg-divert`
- Stat-override: tell dpkg to keep file permissions/owner at some value
  - `dpkg-statoverride`
- These are used by packaging scripts, too. Try listing

$ dpkg-divert –divert /etc/dnsmasq.conf.dpkg –rename /etc/dnsmasq.conf

```
Adding 'local diversion of /etc/dnsmasq.conf to /etc/dnsmasq.conf.
```

$ dpkg-statoverride –add richard richard 0755 /etc/bash.bashrc $ dpkg-divert –list $ dpkg-statoverride –list

equivs

equivs lets you easily make meta-packages.

- Sample uses:
  - Meta-package to pull in programs you use often
  - Fix broken dependency problem (your package Provides: something)
  - Let dpkg know about your local version
- Can include dependencies, scripts, and files
- equivs-control and equivs-build

$ **less rkd-standard-system**

```
...
Package: rkd-standard-system
Version: 1.0
Maintainer: Richard Darst <rkd@zgib.net>
Depends: less,
         screen,
         python2.5,
         bash-completion,
         openssh-server,
         openssh-client
...
```

$ **equivs-build rkd-standard-system**

The alternatives system

Debian has many choices, and the alternatives system allows the user to choose among them.

- Sometimes different packages can provide equivalent utilities.
- Alternates system selects among them
- `/usr/bin/vim` links to `/etc/alternates/vim/` links to `/usr/bin/vim.tiny`
- Different packages install alternates with different priorities
- Select among them with `update-alternatives`. Handles things like man page links also.
- GUI interface in package `galternatives`
- You have to install the relevant packages first!

$ **sudo update-alternatives –list vim**

```
/usr/bin/vim.basic
/usr/bin/vim.basic
/usr/bin/vim.nox
```

$ **sudo update-alternatives –config vim**

dpkg-repack

dpkg-repack undoes an installation in emergency situations.

- Convert installed files back to the package it came from
- includes control info, scripts, and files
- Inherits changes that have been made to the files
- Kind of a hack, use a native package if possible
- rpm, deb, tgz, pkg

alien

You can convert between package formats with `alien`

- Is **not** safe since different OSs have different conventions (e.g. Debian Policy vs Red Hat Policy)
- But for small unobtrusive things it's probably safe enough.
- Do **not** use for major system utilities

auto-apt

auto-apt traces a program to figure out what packages it needs.

- Run a program in auto-apt environment and it will trap system calls for missing files
- Can automatically install packages providing the missing files

Debtags

To help people find what they are looking for, the Debtags system was developed

- Free-form tags for packages
- `http://debtags.alioth.debian.org/ssearch.html` - package search, enhanced by debtags
- `http://debtags.alioth.debian.org/cloud/` - interactive tag cloud
- You can help tag packages online, too!

Kernel help: make-kpkg and module-assistant

There are tools which help create packages of kernel images and modules.

- make-kpkg automates building and packaging kernels when you get the source from elsewhere
- module-assistant automates building module packages
  - The *packagename*-source packages use this
- *packagename*-dkms ("Dynamic Kernel Module Support") packages are another system for automatically compiling modules.

$ **aptitude install lustre-source –without-recommends**
$ **module-assistant auto-install lustre**

reportbug and reportbug-ng

If people help report bugs, Debian's quality goes up.

- http://bugs.debian.org
- There isn't a web interface to *report* bugs
- reportbug and reportbug-ng make reporting easy, though. Gathers all information and submits.

$ **reportbug –email=MYEMAIL@zgib.net python-lzma**

```
Which of the following packages is the bug in?
Just press ENTER to exit reportbug.

1 python-lzma       Python bindings for liblzma

2 python-lzma-dbg  python-lzma debug symbols
Select one of these packages: 1
Please enter the version of the package this report applies to (bl
>
Will send report to Debian (per lsb_release).
Querying Debian BTS for reports on python-lzma (source)...
1 bug report found:

Outstanding bugs -- Minor bugs; Unclassified (1 bug)
  1) #594943  python-lzma has duplicated changelog.gz
Please briefly describe your problem (max. 100 characters allowed;
```

Useful web services

Over the years Debian has developed a lot of useful but undermarketed web services.

- `backports.debian.org` - Find newer versions of packages for stable
- `snapshot.debian.org` - Archive of every single debian package on the mirrors ever
- `bugs.debian.org` - Search bug reports for a package
- `packages.debian.org` - Web interface to some of these tools

Source package layout

Source packages contain buildable source code, and can be retrieved from the `APT` system.

- `debian/` directory should contain all Debian information
  - `debian/rules` - build instructions (a Makefile)
  - `debian/control` - Meta-information
  - `debian/changelog` - Version number and changes
- `dpkg-source` packs and unpacks source trees
- `dpkg-buildpackage` builds binary packages
- `dpkg-deb` packs and unpacks binary packages
- Getting the source to any package
  - Make sure a `deb-src` line is in your `/etc/apt/sources.list`
  - `apt-get source` *packagename*
  - `apt-get build-dep` *packagename*

Useful devscripts

devscripts contains a lot of useful tools for packagers and developers.

- rmadison queries versions of packages in all suites
- debdiff intelligently diffs source or binary packages
- aptitude install devscripts --without-recommends
  - Has a *lot* of recommends

$ **rmadison bash**

```
bash |      3.2-4 |        stable | source, alpha, amd64, ar
bash |      4.1-3 |       testing | source, amd64, armel, hp
bash |      4.1-3 |      unstable | source, alpha, amd64, ar
```

Fast package rebuliding

The consistent build interface means it is easy to rebuild packages:

```
$ apt-get source bash
$ sudo apt-get build-dep bash
$ cd bash-VERSION/
$ debuild
```

There is much, much more

- We've gone over some basic utilities here
- This list is biased towards what I know - there so much that I am missing
- Debian-NYC has workshops which go into more detail about the development side of things
- `http://wiki.debian.org/DebianNYC`

More resources

User resources:

- http://www.debian.org/doc/manuals/reference/ - Debian user guide, highly recommended
- http://www.debian.org/doc/manuals/refcard/refcard.en.pdf - Debian reference card

Development resources:

- http://www.debian.org/doc/developers-reference/ - Developer's reference, useful for creating packages
- http://www.debian.org/doc/debian-policy/ - Debian policy document
- http://www.debian.org/doc/maint-guide/ - New Maintainer's guide, first steps to making a package

The end

Thank you

Presentation permissions

You are free to use this presentation under the 3-clause BSD license.