A crash course in handling deb packages

Topics covered in this presentation

- What is a **deb** package?
- What is dpkg?
- What is APT?
- The **apt** util
- **apt-get**
- **apt-cache**
- **apt-file**
- Fetching and installing packages
- Install vs. upgrade
- Configuration operations
- Non-interactive operations
- A note on dependencies
- Package Dependencies & Relationships
- Querying the local packages DB
- Querying remote repos
- {pre,post}install scripts
- Downloading & building packages from source
- Automating the installation of packages that require inputs
- Removing packages
- dist-upgrade
- Thanks & final note

What is a deb package?

deb is the format, as well as filename extension of the software package format for the Debian Linux distribution and its derivatives. Debian packages are standard UNIX *ar* archives that include two tar archives. One archive holds the control information and another contains the installable data.

What is dpkg?

dpkg is a tool to install, build, remove and manage Debian packages.

dpkg is controlled entirely via command line parameters, which consist of exactly one action and zero or more options. The action parameter tells **dpkg** what to do and options control the behavior of the action in some way.

dpkg maintains some usable information about available packages. The information is divided in three classes: states, selection states and flags.

What is APT?

Advanced Package Tool, or APT, is a free-software user interface that works with core libraries to handle the installation and removal of software on Debian, and Debian-based Linux distributions. APT simplifies the process of managing software by automating the retrieval, configuration and installation of software packages.

In the next slide, we'll review some important APT utils and their respective functionality.

The apt util

apt provides a high-level CLI for the package management system. It is intended as an end user interface and enables some options better suited for interactive usage by default compared to more specialized APT tools like **apt-get(8)** and **apt-cache(8)**.

apt-get

apt-get is the CLI tool for handling packages, and may be considered the user's "back-end" to other tools using APT. Several *front-end* interfaces exist, such as **aptitude(8)**, **synaptic(8)** and **wajig(1)**.

apt-cache

apt-cache performs a variety of operations on APT's package cache. **apt-cache** does not manipulate the state of the system; it provides functionality to search and generate interesting output from the package metadata.

The metadata is acquired the **update** action of *apt-get/aptitude/apt*, so that it can be outdated but in exchange **apt-cache** works independently of the availability of the configured sources; to wit: it will retrieve what data it has stored on the local system, even when offline..

apt-file

apt-file is a command line tool for searching for files in *deb* packages

Fetching and installing packages

As noted in the last slide, there are multiple utils under the *APT* umbrella. If **apt** is not installed, you can either install it (say with **# apt-get install apt**) or simply substitute **apt** with **apt-get** or **aptitude**.

The below command will: - Retrieve all packages **package-name** depends on - Invoke **dpkg** to install all mandatory dependencies and, finally, **package-name** itself - Again using **dpkg**, run the configuration scripts for all needed packages

```
$ sudo apt update && sudo apt install <package-name>
```

NOTE: **apt update** is needed to download package information from all configured sources. Other commands operate on this data to e.g. perform package upgrades

Install vs. upgrade

If the package is already installed and a new version is available, an upgrade will take place.

Configuration operations

As noted previously, the *deb* format supports pre/post installation and de-installation hooks. These hooks are covered in **slide #9**.

As *deb* supports interactively obtaining inputs from the user, one common scenario is wanting to reconfigure a given package. This can be done thusly:

```
$ sudo dpkg-reconfigure tzdata
```

Non-interactive operations

By default, *apt* will prompt for approval before taking any actions. If that's undesirable, pass the **-y** flag when invoking *apt* (e.g: **apt install -y <package-name>**)

Since the deb format also supports obtaining interactive inputs from the user (using the *debconf* mechanism), you may want to automate that as well. We'll cover that in slide #12.

A note on dependencies

The *deb* spec allows you to specify mandatory dependencies (of course) as well recommended and suggested ones. For instance, let's look at the metadata for the **apache2** package. We can do this by invoking:

```
$ apt show apache2

# or:

$ apt-cache show apache2
```

Here's the truncated sample output:

```
Package: apache2
Version: 2.4.57-2
Priority: optional
Section: httpd
Maintainer: Debian Apache Maintainers <debian-apache@lists.debian.org>
Installed-Size: 580 kB
Provides: httpd, httpd-cgi

Pre-Depends: init-system-helpers (>= 1.54~)
Depends: apache2-bin (= 2.4.57-2), apache2-data (= 2.4.57-2), apache2-utils (= 2.4.57-2),
lsb-base, media-types, perl:any, procps
Recommends: ssl-cert
Suggests: apache2-doc, apache2-suexec-pristine | apache2-suexec-custom, www-browser
```

In the next slide, we'll elaborate as to what Pre-Depends, Depends, Recommends and Suggests indicate.

Package Dependencies & Relationships

Packages can declare that they have certain relationships to other packages.

This is done using the Depends, Pre-Depends, Recommends, Suggests, Enhances and Conflicts control fields.

- Depends: declares an absolute dependency. A package will not be configured unless all of the packages listed in its *Depends* field have been correctly configured
- Pre-Depends: This field is like *Depends*, except that it also forces dpkg to complete installation of the packages named before even starting the installation of the package which declares the pre-dependency. This is important when the configuration phase of one package depends on the other having been configured
- Recommends: This declares a strong, but not absolute, dependency.
- Suggests: This is used to declare that one package may be more useful with one or more others.
- Enhances: This field is similar to *Suggests*. It is used to declare that a package can enhance the functionality of another package.
- Conflicts: When one binary package declares a conflict with another using a Conflicts field, dpkg will refuse to allow them to be unpacked on the system at the same time.

For more info, see **Declaring relationships between packages**

Querying the local packages DB

One of the clear advantages of using packages managers is having a DB you can query to ascertain what's installed on your system, of what version, where the files reside, their nature (config, doc, binary, etc) and what package owns what file. Below are some common queries one might want to make...

Output all installed packages and their respective size:

```
$ dpkg-query -Wf '${Installed-Size}\t${Package}\n' | sort -n
```

Output config files for package:

```
$ dpkg-query -W -f='${Conffiles}\n' bash
```

Sample output:

```
 /etc/bash.bashrc 89269e1298235f1b12b4c16e4065ad0d
 /etc/skel/.bash_logout 22bfb8c1dd94b5f3813a2b25da67463f
```

List packages whose name contains *gcc*:

```
$ dpkg -l "*gcc*"
```

```
||/ Name       Version       Architecture Description
+++-==========================================================
ii  gcc        4:10.2.1-1    amd64        GNU C compiler
ii  gcc-10     10.2.1-6      amd64        GNU C compiler
```

List all files belonging to a given package:

```
$ dpkg -L llvm-11-dev
```

Output the package name to which a file belongs:

```
$ dpkg -S /etc/issue
```

Output the status for a given package (see **man dpkg** for possible states):

```
$ dpkg -s mdp |grep Status
```

Querying remote repos

We've seen how we can easily query our machine's DB for package related data. Now, let's see some examples of querying the remote repos.

Say you're compiling a project and the configure phase fails with this message:

../include/viewer.h:41:10: **fatal error**: ncurses.h: No such file or directory **41 | #include *<ncurses.h>**

Clearly, we're missing the *ncurses* header file but... How can we tell which package provides it? Well, we can use **apt-file** (note: you may need to install it first):

```
$ sudo apt-file update && apt-file search "ncurses.h"
```

On my system, this outputs:

```
libncurses-dev: /usr/include/ncurses.h
libncurses-dev: /usr/include/ncursesw/ncurses.h
```

Note: **apt-file search** does not require super user privileges but because we need to update the DB first, I've used **sudo** for updating.

Another common query is to check what versions of a given package are available. For that, we can use the **apt-cache** util:

```
$ apt-cache policy gcc
```

This will output the available versions for the **gcc** package, as well as the repos they reside in. For example:

```
gcc:
  Installed: 4:10.2.1-1
  Candidate: 4:12.3.0-1
  Version table:
     4:12.3.0-1 500
        500 http://deb.debian.org/debian testing/main amd64 Packages
 *** 4:10.2.1-1 500
        500 http://deb.debian.org/debian bullseye/main amd64 Packages
        100 /var/lib/dpkg/status
```

{pre,post}install scripts

```
$ ls -al /var/lib/dpkg/info/apache2*inst*
```

The above command will output the pre/post-install config scripts invoked when install the **apache2** package. These will typically be shell scripts (though it's not a requirement and any language can be used). *dpkg* will run these scripts with certain arguments, depending on the operation that needs to run.

Usage summary for a *postinst* script:

- **configure**
- **abort-upgrade**
- **abort-remove in-favour**
- **abort-deconfigure in-favour removing**

{pre,post}rm scripts

Similarly, there are removal hooks, which you can locate thusly:

```
$ ls -al /var/lib/dpkg/info/apache2*rm*
```

Usage summary for a *postrm* script:

- `remove'
- `purge'
- `upgrade'
- `failed-upgrade'
- `abort-install'
- `abort-install'
- `abort-upgrade'
- `disappear'

All hooks can be edited and run manually. And indeed, there will be times when you'll want to do just that; to wit: when they failed, leaving your package in a half-baked state. In such a case, you can locate the problematic code, correct it and invoke **apt install -f** or **dpkg-reconfigure <package-name>** to allow **dpkg** to finish the deployment.

(For more details, see **the Debian docs** or the debian-policy package)

Downloading & building packages from source

Some use-cases:

- Review the patches the distro maintainer applied to the pristine source
- Patch the source and build your own, custom package
- Shamelessly steal some of the specs as you're creating a similar package :)

To download the source package:

```
$ apt-get source <package-name>
```

Here are some important files you'll find under the resulting **<package-name>-<package-version>/debian dir**:

- debian/changelog: as the name implies, this will include the changes the maintainer made per release (you can get that bit with **apt changelog <package-name>** as well)
- debian/control: metadata for the package (version, deps, description, maintainer, section, priority, etc)
- debian/rules: the build instructions (this will typically be a **Makefile** but it can be any script)
- debian/patches: patches applied to the pristine source

Before building, make sure you've installed the **build-essential** package and invoke the below command to install needed build deps (as declared in debian/control):

```
$ sudo apt-get build-dep <package-name>
```

Finally, to build the package invoke:

```
$ dpkg-buildpackage -b -uc
```

Note on **build-essential**: This package is a meta package that depends on a list of packages which are considered essential for building Debian packages. On my Debian 11, these are:

```
Depends: libc6-dev | libc-dev, gcc (>= 4:12.3), g++ (>= 4:12.3), make, dpkg-dev (>= 1.17.11)
```

If you intend to get serious about building debs, installing **devscripts** is also a good idea. Your package may, of course, have additional deps. To review them, see the *Build-Depends* section in *debian/control*

For more details, see **Debian's building tutorial**.

Automating the installation of packages that require inputs

As mentioned in a previous slide, some packages require user inputs. By default, these are obtained interactively. Of course, when automating the installation of a package, this must be handled differently.

First, install the package manually. We'll use the *tzdata* package as an example (chances are, it's already installed on your machine).

To get the mandatory inputs, invoke:

```
$ debconf-show tzdata|grep "^*"
```

From these, you'll need to create a response file, for example:

```
tzdata tzdata/Areas select Europe
tzdata tzdata/Zones/Europe select London
tzdata tzdata/Zones/Etc select UTC
```

Then, feed this to **debconf-set-selections**, thusly:

```
$ sudo debconf-set-selections /path/to/tzdata/response/file
```

To see that it was applied, invoke **debconf-show tzdata** once more.

And finally, run:

```
$ sudo dpkg-reconfigure -f noninteractive tzdata
```

Note I: **tzdata** will only require these inputs if no **/etc/timezone** exists (see **/var/lib/dpkg/info/tzdata.postinst**) so, in order to properly test this worked, you'll need to remove this file.

Note II: the **dpkg-reconfigure** command above is good for testing purposes, for the actual deployment, you'll want to **export DEBIAN_FRONTEND=noninteractive** in your script so that you're not prompted for any inputs whilst installing packages.

dist-upgrade

dist-upgrade in addition to performing the function of upgrade, intelligently handles changing dependencies with new versions of packages; **apt-get** has a "smart" conflict resolution system, and it will attempt to upgrade the most important packages at the expense of less important ones if necessary.

The dist-upgrade command may therefore remove some packages. The /etc/apt/sources.list file contains a list of locations from which to retrieve desired package files. See also **apt_preferences(5)** for a mechanism for overriding the general settings for individual packages.

Removing packages

APT supports several removal operations; these are:

- remove: will remove a package but leave its configuration files on the system
- purge: identical to remove except that any configuration files are deleted too
- autoremove: used to remove packages that were automatically installed to satisfy dependencies for other packages and are now no longer needed

Normally, I'd advise using **apt remove <package-name>** as the configuration files may come in handy, For instance, let's say you're removing the **mariadb-server** package because you want to run your own, custom version of it. The config files you've laboured on will still be needed and there's no reason to recreate them.

apt purge <package-name> is handy if your package is malfunctioning and you want to start anew. In such a case, you may also want to inspect your past **debconf** selections (and potentially altering them) before giving it another go.

To see the difference in action:

- List all **<package-name>** config files with: **dpkg-query -W -f='${Conffiles}\n' <package-name>**
- Run **apt remove <package-name>**
- Conf files should still be on your FS
- Run **apt purge <package-name>**
- Conf files will no longer be present

Lastly, **apt autoremove** requires no arguments as its job is to get rid of all unneeded packages. This is handy for clearing space, as well as keeping your system lean. You know what they say... Less is more:)

My sincere thanks to...

- The hard working people at **Debian GNU/Linux**
- The fine people who contributed to **mdp** (used to create this presentation)
- Marcin Kulik for creating **ASCIInema** (and everyone's who's contributed to it since)

Resources for this course can be found at my **Crash course in...** repo. Contributions (as well as shares) are welcomed.

Final note

Psst.. Liked this content and have a role I could be a good fit for? I'm open to suggestions. See **https://packman.io/#contact** for ways to contact me.

Cheers, Jesse