



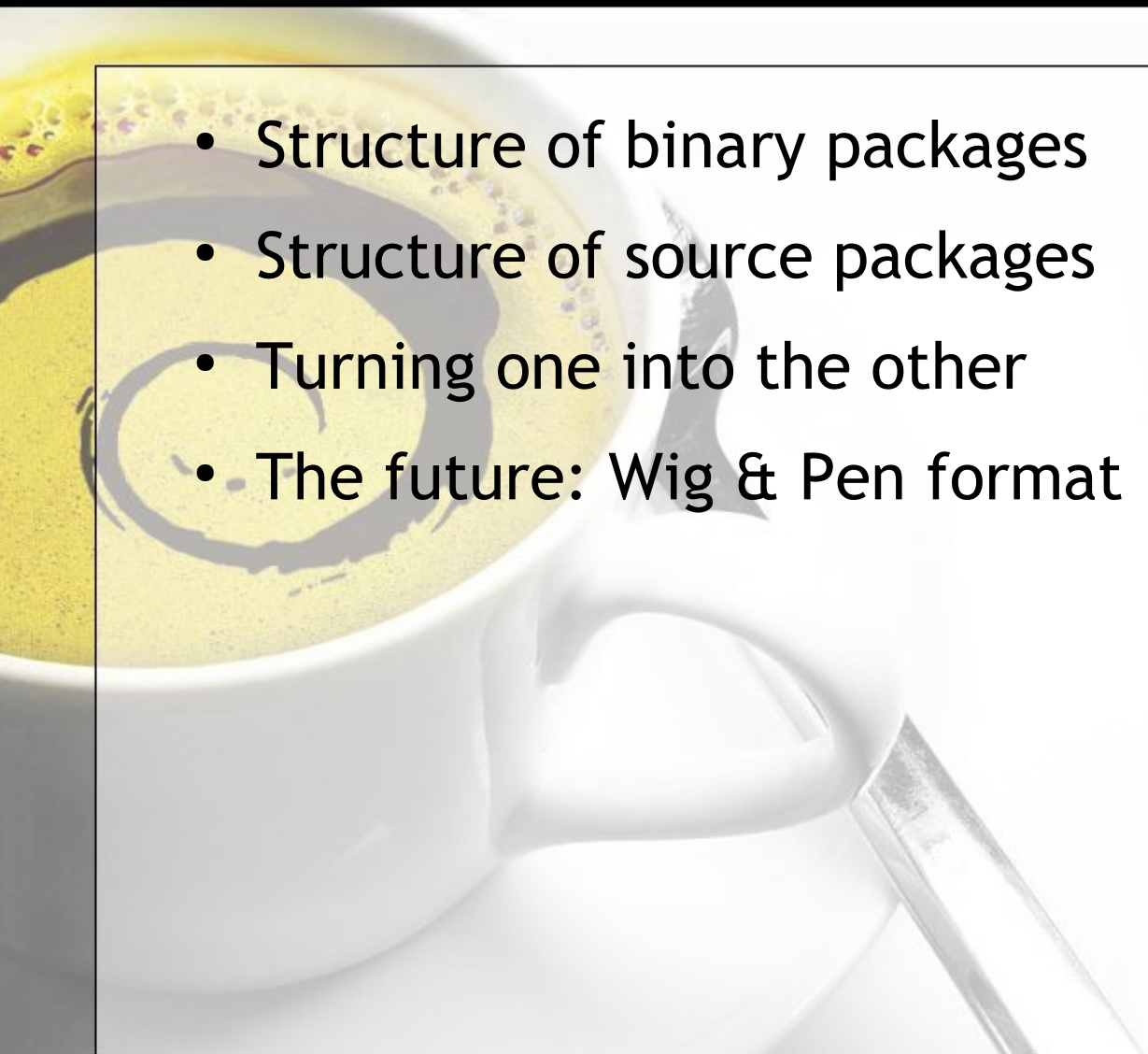
Anatomy of a Debian Package

Jonathan Oxe, Internet Vision Technologies
jon@debian.org / jon@ivt.com.au

Google Tech Talk
July 21st 2006

debian

There And Back Again

- 
- A close-up photograph of a white ceramic coffee cup filled with a yellowish-brown latte. The surface of the coffee is decorated with a dark brown latte art design that resembles a bird in flight, with its wings spread. A silver spoon is partially visible, resting on the white saucer beneath the cup. The background is a soft, out-of-focus light gray.
- Structure of binary packages
 - Structure of source packages
 - Turning one into the other
 - The future: Wig & Pen format

Binary Package Format

Every “.deb” package is actually just an “ar” archive containing:

- **data.tar.gz** (files for the filesystem)
- **control.tar.gz** (maintainer scripts and extra meta data)
- **debian-binary** (package version: currently 2.0)

You can therefore manually create and manipulate Debian packages using standard Unix tools if you feel particularly masochistic.

Examining Binary Packages

Or do it the easy way with the tools designed for the purpose:

- **`dpkg --info foo.deb`** - Examine package metadata
- **`dpkg --contents foo.deb`** - File listing
- **`dpkg --unpack foo.deb`** - Extract the package locally
- **`dpkg --install foo.deb`** - Extract and run configure scripts
- **`dpkg --help`** - A freaky number of options

Package Metadata

Binary package metadata viewable with **dpkg --info** includes:

- **Package:** Actual package name. Filename is unimportant.
- **Source:** The source package that this binary package was built from.
- **Version:** The full package version, including the upstream version (before the hyphen) and Debian version (after the hyphen).
- **Architecture:** What CPU the package was built for.
- **Depends, Recommends, Suggests, Replaces, Conflicts, Enhances:** Relationships with other packages.

Maintainer Scripts

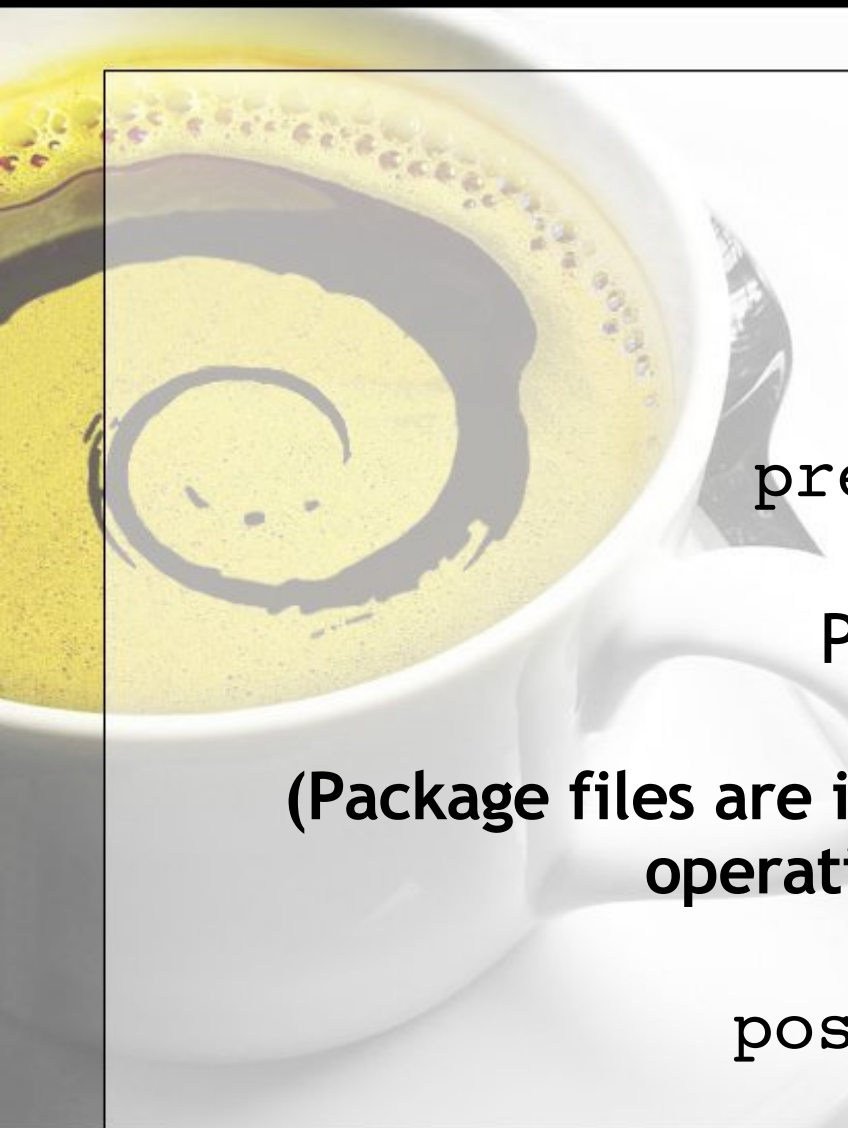
Maintainer scripts are executed on package installation and removal. Typically written in BASH or Perl. The standard scripts are:

- **preinst**: Run prior to extraction onto filesystem.
- **postinst**: Run after extraction onto filesystem.
- **prerm**: Run prior to removal from filesystem.
- **postrm**: Run after removal from filesystem.

To standardise and abstract configuration you can also use:

- **config**: Ask questions to seed to other maintainer scripts.

Installation Sequence



```
dpkg -i
```



(Debconf)



```
preinst install
```



Package unpack



(Package files are installed, but config necessary for operation hasn't been done)



```
postinst install
```



(Package is fully installed and ready to use)

Removal Sequence

`dpkg -r`



`prerm remove`



(Program files are removed)



`postrm remove`



(Residual config information is all that remains)



`postrm purge`



(Nothing left)

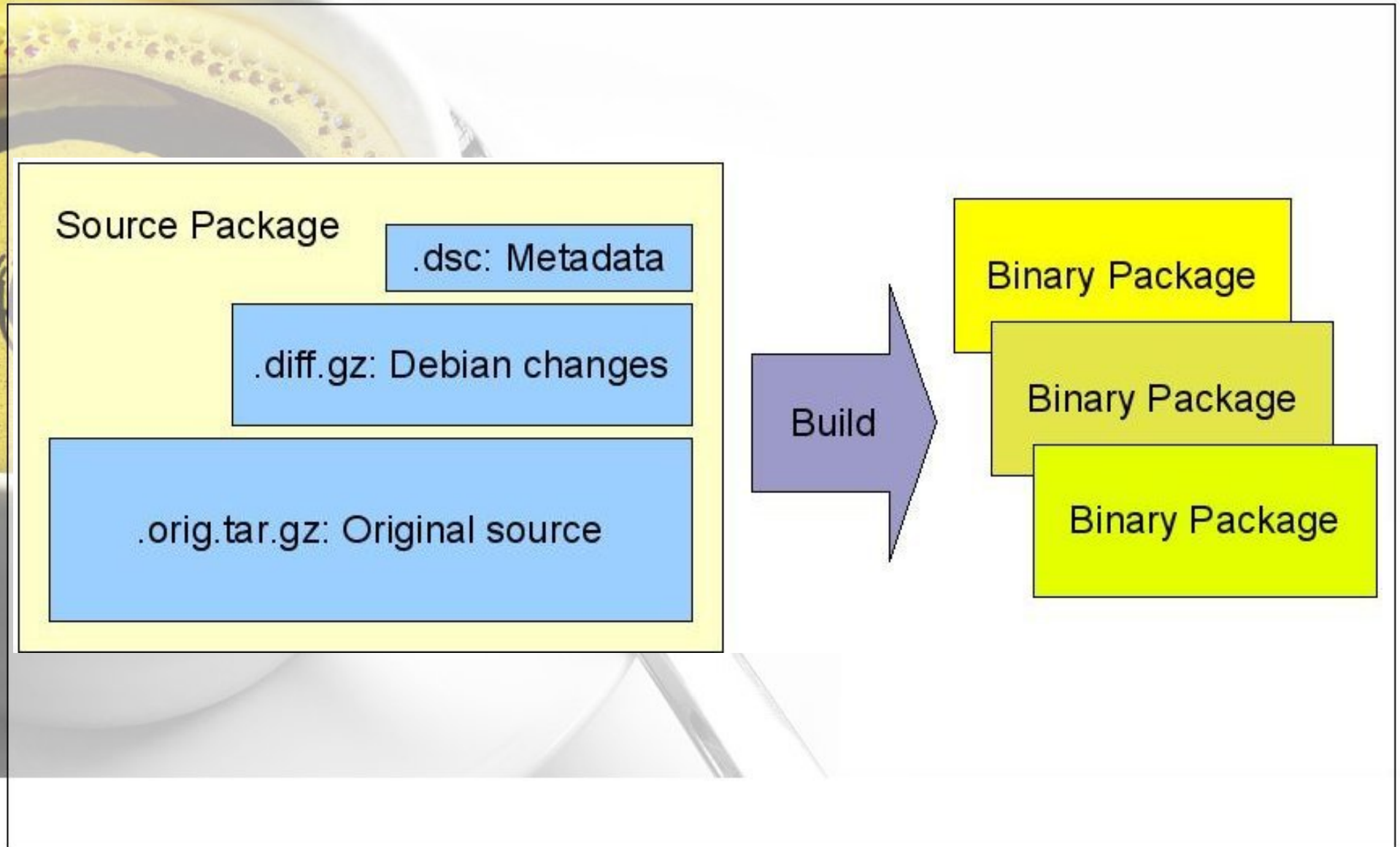
Source Package Format

A source package is an architecture-independent archive of everything needed to build a binary package:

- ***foo.dsc***: Overall control file containing a description and fields for build parameters.
- ***foo.orig.tar.gz***: Original source code as provided by upstream (usually).
- ***foo.diff.gz***: All changes applied for Debian.

“Native” packages have no orig/diff split, and instead have everything in a single tarball. Don't use this format for normal packages.

Package Diagram



A Source Package Unpacked

Most changes in a Debianised source package are localised in the **debian** directory which contains:

- Metadata to control the build process.
- Maintainer scripts.

Debian has strict policy and often changes are needed directly in the source tree itself, but follow the “minimal impact” principle to save yourself headaches. Pass as many changes upstream as possible.

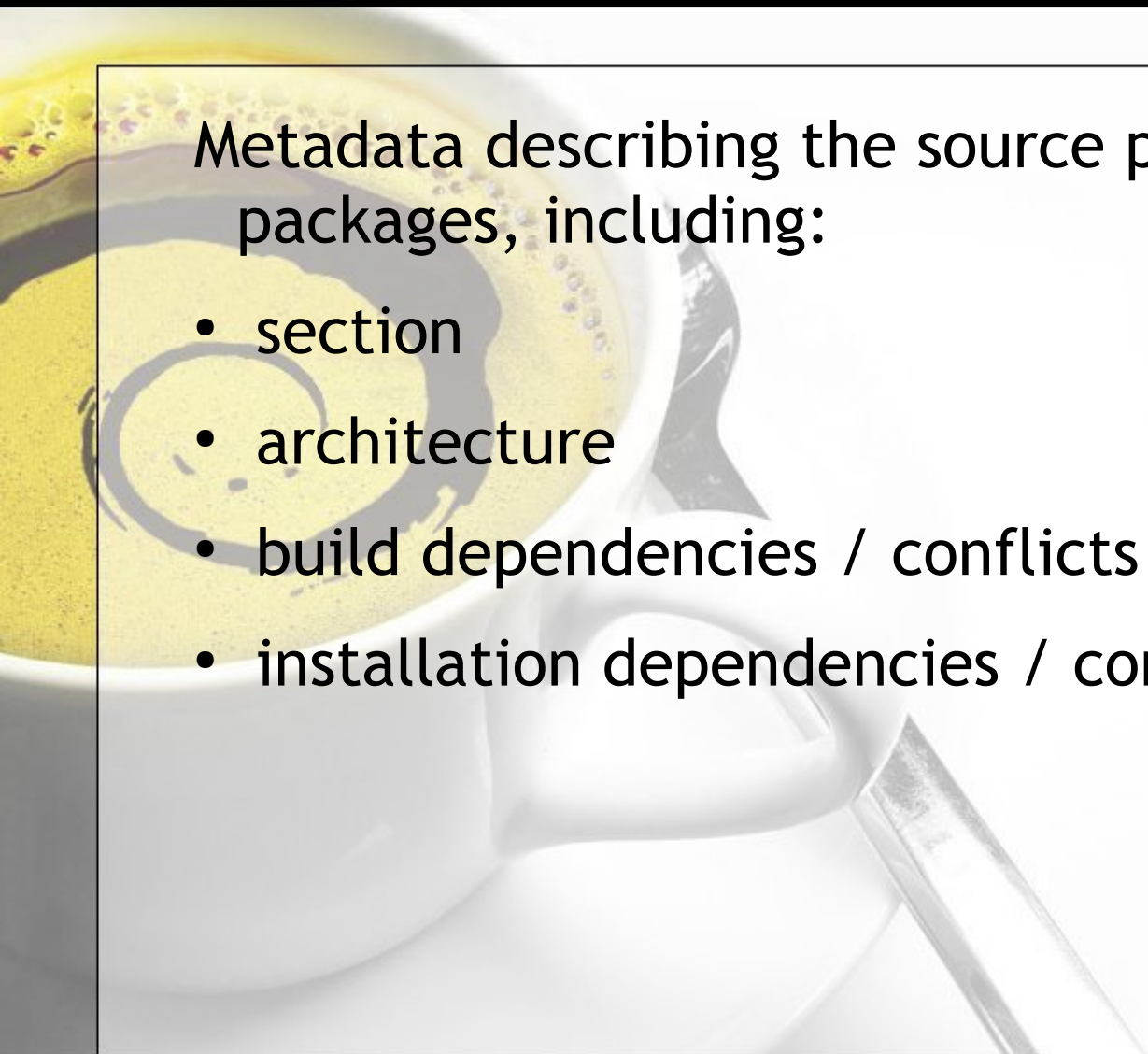
You will probably need to change some paths though to make the final package policy-compliant. No “/usr/local”!

debian/rules

debian/rules controls how the package is built. Typically a makefile with multiple targets corresponding to various stages of the build process:

- **configure**: Does pre-build configuration such as running `./configure` with appropriate options. (optional)
- **build**: Compiles the package from source. (required)
- **install**: copies / moves files from their build destination into the installation tree. (optional)
- **binary**, **binary-arch**, and **binary-indep**: Create the binary packages. Typically `binary` calls `-arch` and `-indep`.
- **clean**: Returns the package to the pre-build state.

debian/control

A close-up photograph of a white ceramic coffee cup filled with a yellowish-brown latte. The surface of the coffee has a dark brown, swirling latte art design. A silver spoon is partially visible, resting on the rim of the cup. The background is a soft, out-of-focus white.

Metadata describing the source package and all binary packages, including:

- section
- architecture
- build dependencies / conflicts
- installation dependencies / conflicts

Building A Package

Minimum requirement is that the binary target of `debian/rules` must produce the binary packages specified in `debian/control`. In principle, all you need to do is change into the root of the package source tree and run:

```
debian/rules binary
```

as root. That leaves pretty much everything up to the package maintainer when writing the `debian/rules` file, and can result in lots of shell script fragments doing sick and twisted things.

Avoid the anarchy by using a build helper system.

The debhelper Build Helper

The most commonly used build help system is debhelper, which provides a swag of `dh_*` commands for all sorts of common tasks.

```
apt-get install debhelper dh-make
```

Other useful debhelper components include `dh-make-perl` and `dh-make-php`.

Bootstrap your package by extracting and entering the source tree, then:

```
dh_make -f ../foo-0.1.tar.gz -e you@example.com
```

Build-time Helpers

Don't just run the build scripts directly: use one of the several very useful build process scripts to automate the `debian/rules` binary call and associated scaffolding.

- **dpkg-buildpackage**: Cleans source, builds the binary packages, builds description and changes files, and signs the package ready for upload.
- **debuild**: Wraps `dpkg-buildpackage` and adds extra bits like automatic lintian / linda checks.
- **pbuilder**: Builds in a clean chroot environment.
- **cvs-buildpackage**: Pulls a release from CVS and builds it. There are also **svn-** and **tla-buildpackage** equivalents.

Other Build Helpers

A close-up photograph of a white ceramic coffee cup filled with a light-colored latte. The surface of the coffee is decorated with a dark brown, swirling latte art design. A silver spoon is partially visible, resting on the white saucer beneath the cup. The background is a soft, out-of-focus light gray.

Creating your own build helper suite is a popular sport for Debian developers. Alternatives include:

- dpatch
- dbs
- cdbb

and many others.

Testing with Lintian and Linda

Many common packaging mistakes and policy violations can be caught by Lintian and Linda, the two main package sanity-check tools:

```
apt-get install lintian linda
```

```
lintian foo_1.0.1_i386.changes
```

```
linda foo_1.0.1_i386.changes
```

Publish Your Packages

Install packages locally with `dpkg` or publish them on a package server (just an HTTP or FTP server) with a package index:

```
apt-get install apt-utils  
mkdir -p /var/www/ubuntu/dapper  
cp foo-1.0.1_i386.deb /var/www/ubuntu/dapper/  
cd /var/www/ubuntu  
apt-ftparchive packages dapper > dapper/Packages  
gzip dapper/Packages
```

On your client machine add to `/etc/apt/sources.list`:

```
deb http://www.example.com/ubuntu dapper/
```

Updates

Version numbers and changelog entries are set in `debian/changelog`.

Increment it using:

```
dch -i
```

`dch` is part of the `devscripts` package which contains a lot of useful scripts to make your life easier:

```
apt-get install devscripts
```

The “Wig & Pen” Format



The “Wig & Pen” format (ie: source format v 2.0) is an evolutionary rather than revolutionary change to the current source package format.

- Multiple upstream tarballs supported.
- The “Debian Diff” may be replaced by a “Debian Tar”.
- Bzip2 compression supported as alternative to gzip.

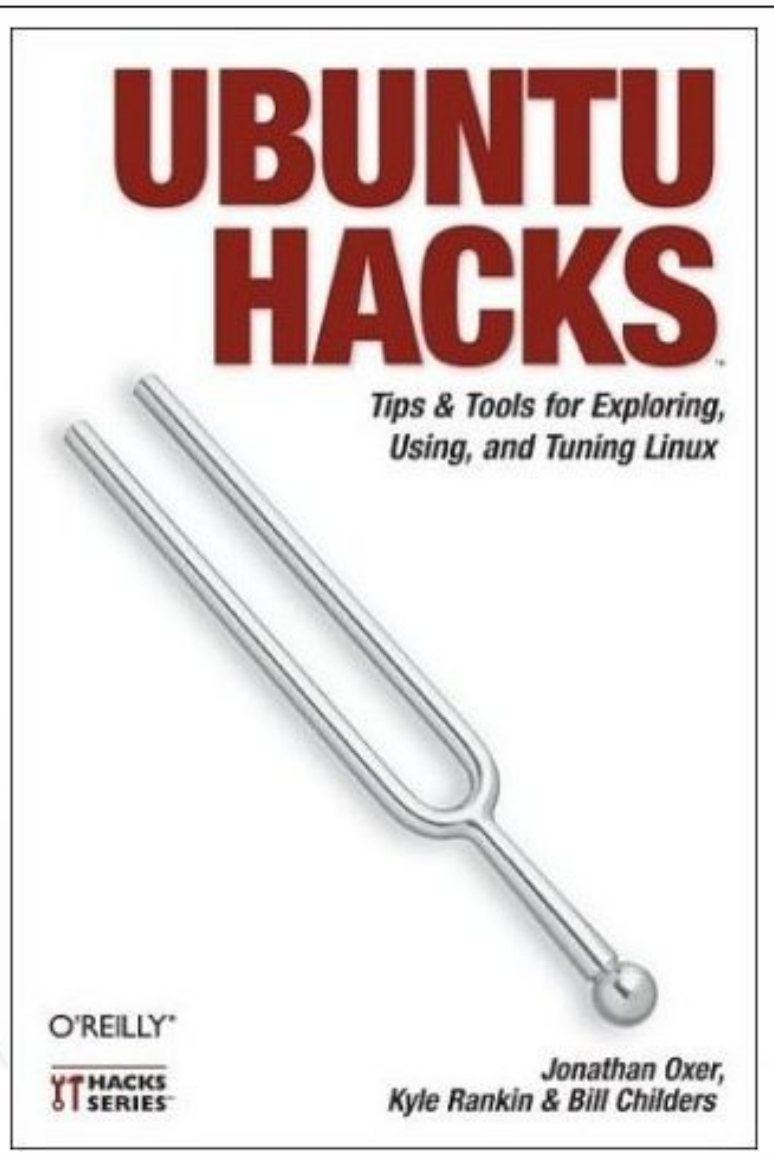
Blatant Advertising

Ubuntu Hacks: Tips & Tools for Exploring, Using, and Tuning Linux

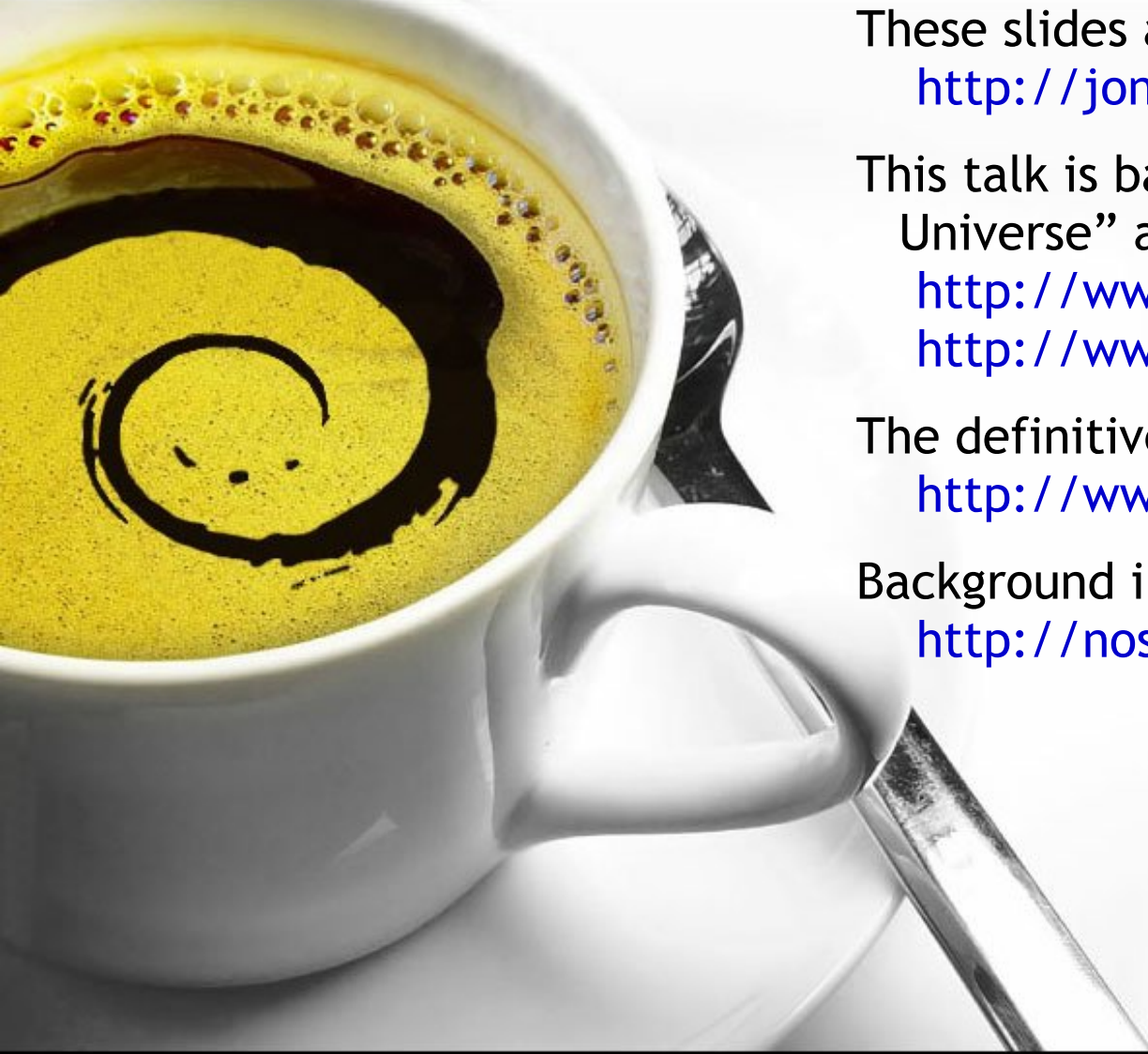
Buy a copy! Buy two! Tell all your friends!

Includes info on:

- Converting RPM to DEB
- Rebuilding source packages
- Packaging kernels
- Package caching



More Information



These slides are online at:

<http://jon.oxer.com.au/talks/>

This talk is based on sections of “The Debian Universe” and “Ubuntu Hacks”:

<http://www.debianuniverse.com/>

<http://www.ubuntuhacks.com/>

The definitive package maintenance guide:

<http://www.debian.org/doc/maint-guide/>

Background image by Andre Gonzalez:

<http://nostromo2k3.deviantart.com/gallery/>

The Debian logo, which consists of the word "debian" in a lowercase, bold, sans-serif font. A small red diamond is positioned above the letter 'i'.