Debian and Ubuntu Software Packaging Workshop

Alexander Dutton

Developer, Oxford University Computing Services

@alexsdutton – alexander.dutton@oucs.ox.ac.uk

Aims

- A gentle introduction to packaging software for Debian and Ubuntu
  - I am not an expert
  - I've only recently started doing this stuff
- Focus on Python
  - Because that's what I know
  - Notwithstanding some generic usefulness

We'll look at...

- Producing a simple package
- What all those files (e.g. debian/control, debian/compat) mean, and what should be in them
- Packaging Python things in particular
- Overriding some of the package build and installation steps
- Setting up an apt repository and publishing packages
- Things that might catch you out.

Why?

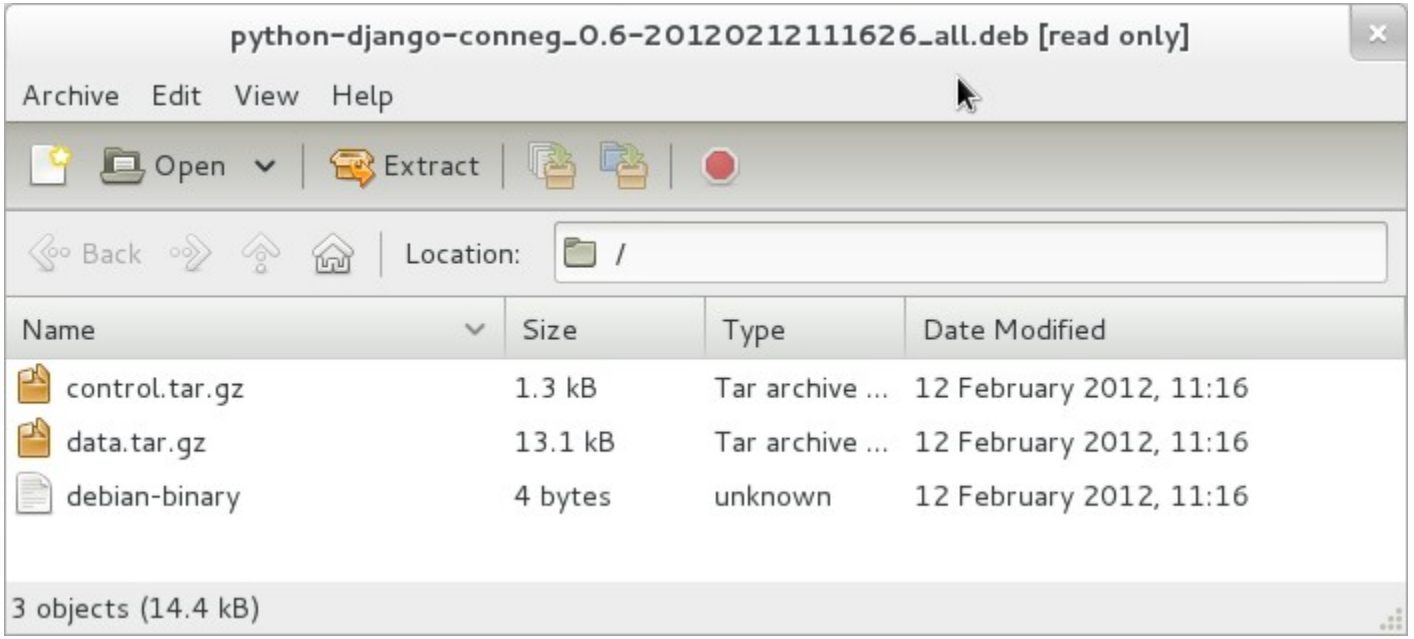
- Reproducible deployments
  - Less bespoke
  - Easier to test updates
  - Bring colleagues up to speed more quickly
- Improved understanding of the Linux/Debian way of doing things
- CV points

The workshop bit

- If you've got your own installation, awesome.
- Otherwise, grab a piece of paper and log in to the box provided.
- This talk is up at https://github.com/alexsdutton/packaging-workshop

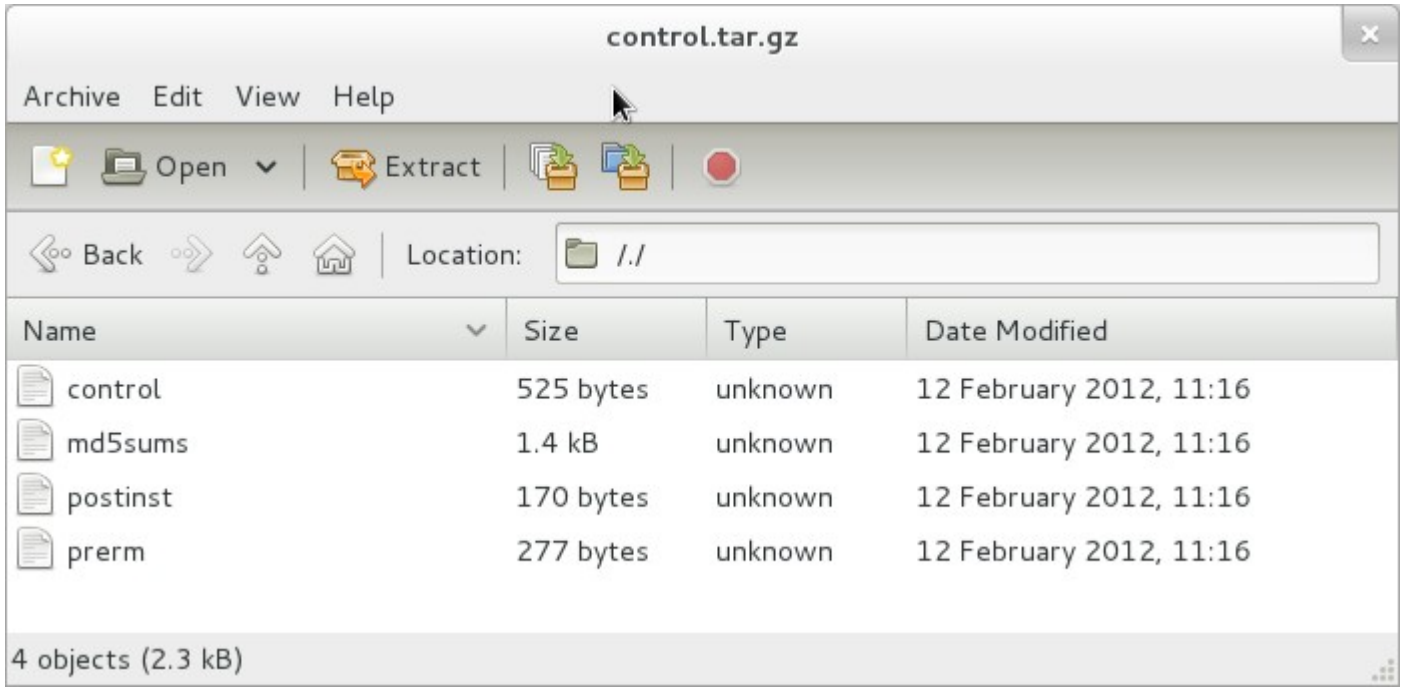
What is a Debian package?

- A .deb is just an **ar** archive
- Containing two .tar.gz archives and a version number:



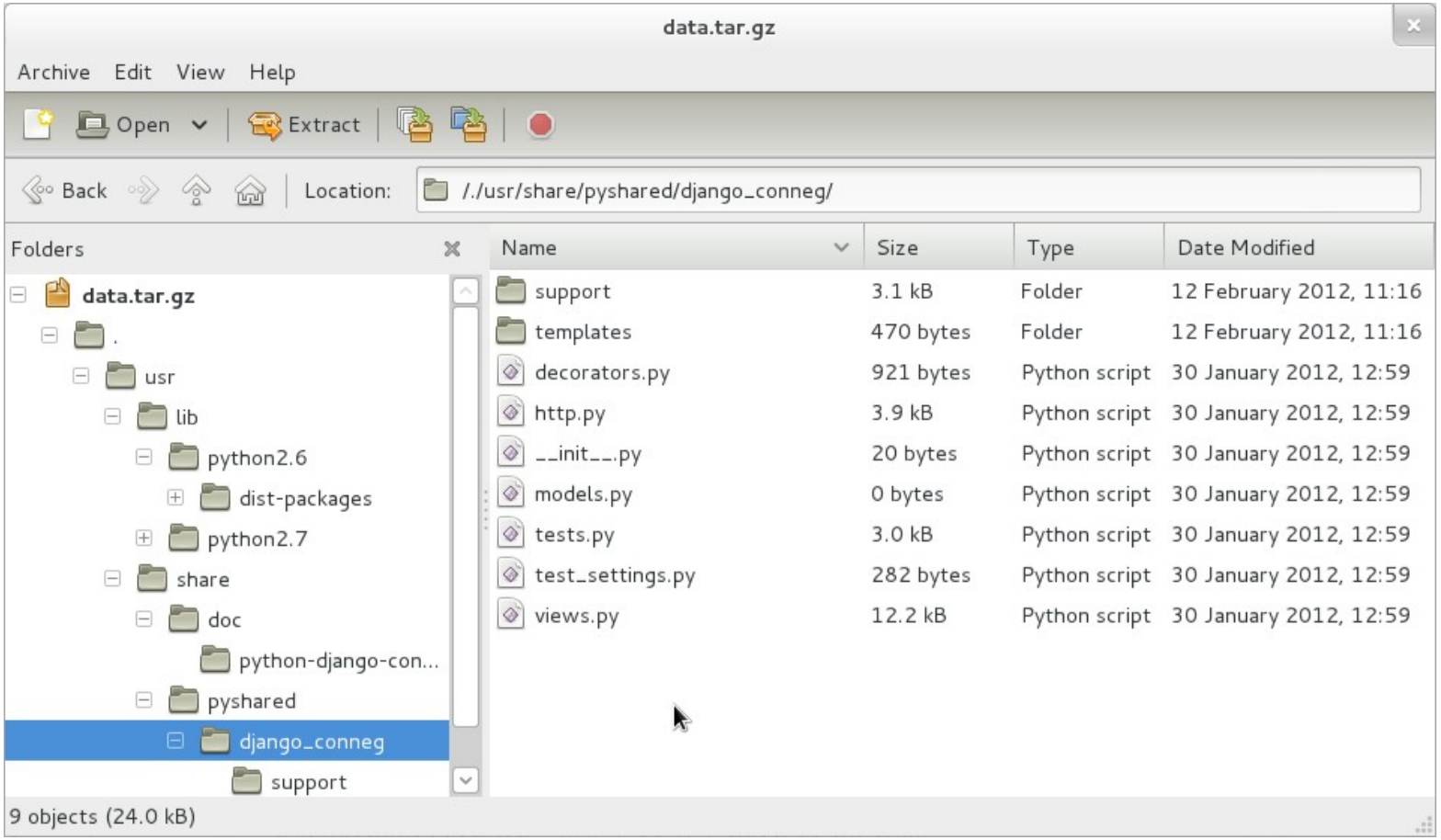
control.tar.gz

- Package metadata:



data.tar.gz

- Extracted onto filesystem when installed



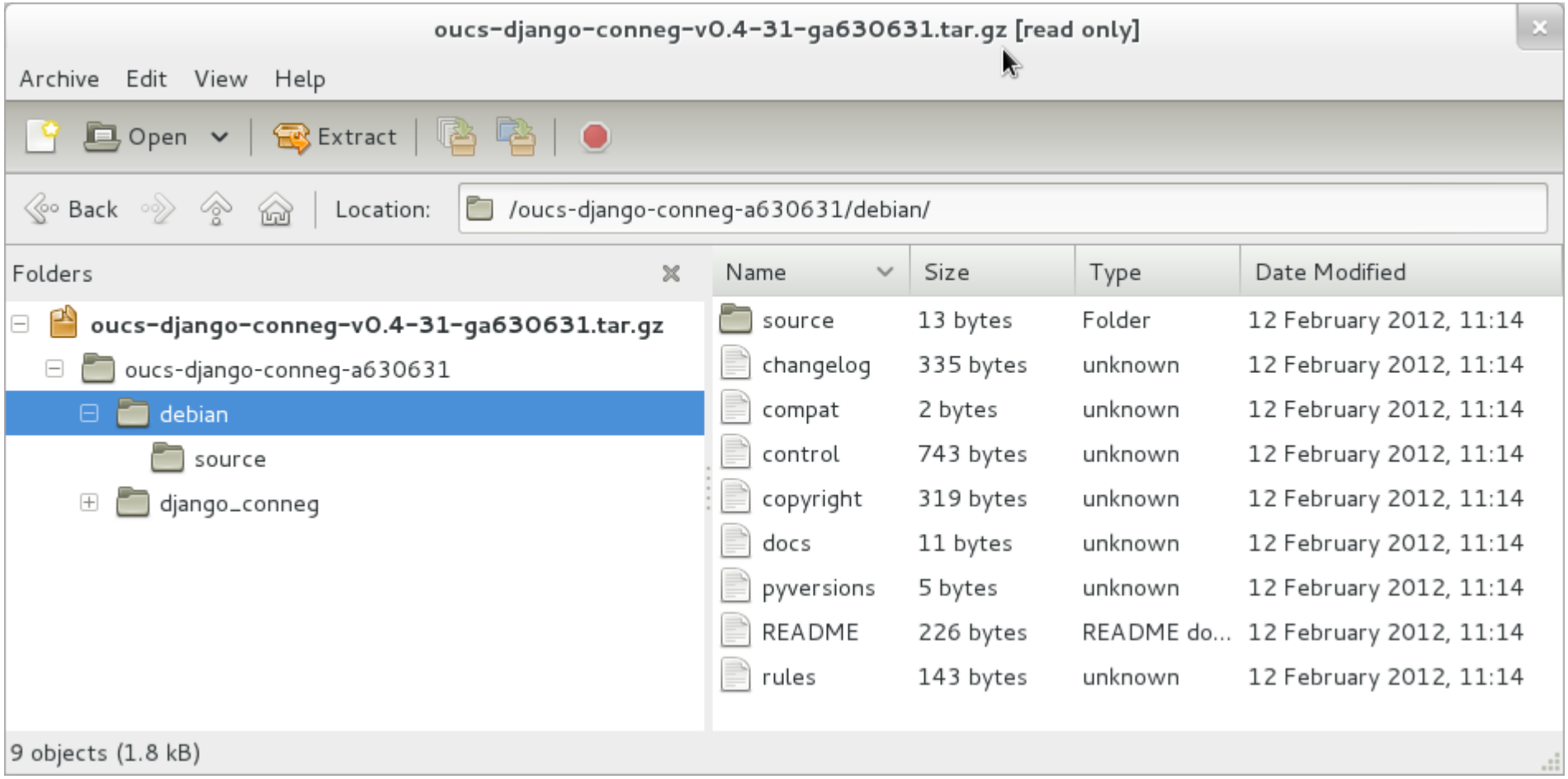
So how do we get there?

- You could build it by hand
  - But that probably won't scale
  - And you'll reinvent lots of wheels along the way
- Instead, use the Debian packaging toolchain

Caveats

- Ignoring non-native packages for now
- This is a little Python-centric
  - Shout if I've assumed too much.

Layout of a native package



(tarball downloaded from GitHub at https://github.com/oucs/django-conneg/downloads)

debian/

- Needs to contain:
  - control (metadata; dependencies; description)
  - changelog (lists changes for each released version)
  - copyright (license information in structured format)
  - rules (a makefile; mostly delegating to debhelper)
  - source/format
    - For us, contains “3.0 (native)”
    - For non-native, should contain “3.0 (quilt)”

What happens

- Call dpkg-buildpackage
- **`debian/rules`** is used to generate the package layout in a directory with the same name as the package
  - in our case, **`debian/python-django-conneg`**
- This all gets wrapped up into a **`.deb`** which is deposited in the parent directory
- Optionally sign using GPG, install, or push to a repository

debian/control

- Contains metadata about the package
  - build dependencies
  - install dependencies
  - package names
  - description
  - targeted architectures
  - suggested packages
- Used by tools when building your package
- http://www.debian.org/doc/manuals/maint-guide/dreq.en.html#control

debian/control

```
Source: python-django-conneg
Section: python
Priority: extra
Maintainer: Alexander Dutton <alexander.dutton@oucs.ox.ac.uk>
Build-Depends: debhelper (>= 7.0.50~),
               python-all,
               python-support,
               python-django (>= 1.3)
Standards-Version: 3.9.1
X-Python-Version: >= 2.6

Package: python-django-conneg
Section: python
Architecture: all
Depends: ${misc:Depends},
         ${python:Depends},
         python-django (>= 1.3)
Description: Class-based views for returning content-negotiated responses
 django-conneg provides a simple and extensible framework for producing
 views that content-negotiate in Django.
```

Source package name

You

Packages that need to be installed to build this package

Source pkg. details

Binary package name (the one that gets installed)

Either 'all' (arch independent), 'any' (arch dependent; build for all), or a list of architectures (e.g. 'amd64, i386')

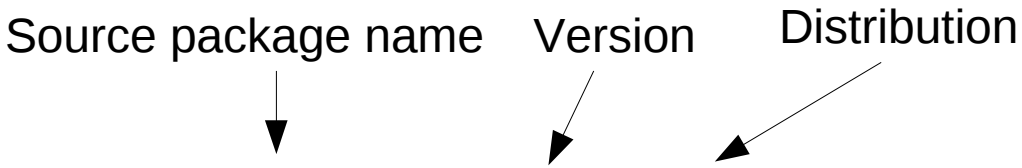
Installed dependencies

Short description (first line)

Long description (indented one space)

Binary pkg. details

debian/changelog




```
python-django-conneg (0.6) unstable; urgency=low

  * Fixed renderer priority handling

 -- Alexander Dutton <alexander.dutton@oucs.ox.ac.uk>  Thu, 26 Jan 2012 09:31:38 +0000

python-django-conneg (0.5) unstable; urgency=low

  * Initial Release.

 -- Alexander Dutton <alexander.dutton@oucs.ox.ac.uk>  Mon, 19 Dec 2011 14:14:33 +0000
```

Description of changes

- Very particular format (see link below)
- Used to obtain version number
- Best to use `dch` to create (and `dch --create` for a new package)
- http://www.debian.org/doc/manuals/maint-guide/dreq.en.html#changelog

debian/copyright

```
Format-Specification: http://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Contact: Oxford University Computing Services <infodev@oucs.ox.ac.uk>
Source: https://github.com/oucs/django-conneg

Files: *
Copyright: 2012 University of Oxford <infodev@oucs.ox.ac.uk>
License: BSD
```

- This one is simple
- http://dep.debian.net/deps/dep5/ has the low-down on how these things are written
- Required if you want to get your packages into Debian. If internal, te amount you care is up to you.
- http://www.debian.org/doc/manuals/maint-guide/dreq.en.html#copyright

debian/rules

```
#!/usr/bin/make -f

%:
	dh $@ --with python2

override_dh_auto_test:
	django-admin test --settings=django_conneg.test_settings --pythonpath=.
```

- A makefile!
- Most targets delegated to **dh** (debhelper)
- Various helper addons available
  - Run **dh -l** for a list of those installed
- http://www.debian.org/doc/manuals/maint-guide/dreq.en.html#rules

Makefiles

- Target names
  - Not indented; suffixed with a colon
- Commands
  - Almost the same syntax as shell commands
  - Indented with a single tab
- Normally invoked as e.g. **`make install`**
  - Invokes install target

```
#!/usr/bin/make -f

%:
	dh $@ --with python2

override_dh_auto_test:
	django-admin test --settings=django_conneg.test_settings --pythonpath=.
```

Overriding things in debian/rules

- debhelper makes it relatively easy to override stages of the package creation process
- See **`man debhelper`** for a list of stages
- Prefix "**`override_`**" to override; possibly call the original afterwards

Overriding example: symlinks

- `dh_link` creates symlinks in the installed package
  - uses a file called `debian/<package>.links`
- We can override it with a `override_dh_link` target
- Silly example (duplicating functionality of dh_link):

```
#!/usr/bin/make -f

%:
	dh $@ --with python2

override_dh_auto_test:
	django-admin test --settings=django_conneg.test_settings --pythonpath=.

override_dh_link:
	# Link to jQuery provided by libjs-jquery package
	ln -s ../../../../libjs/jquery/jquery-min.js \
	  debian/python-django-conneg/usr/share/pyshared/django_conneg/static/js/jquery.js
	# Call the original dh_link to process debian/python-django-conneg.links
	dh_link
```

Other interesting files

- `debian/<package>.<something>` is a theme
- `debian/<package>.install` used to copy files into the target tree
  - <source pattern> [whitespace] <target directory>, e.g. `conf/* etc/mypackage/`
- `debian/<package>.dirs` used to create empty directories
- `debian/<package>.init` installed as initscript in /etc/init.d/
- `debian/{pre,post}{inst,rm}`
  - Scripts which are to be run before or after installation or removal
  - Useful for doing things like adding users or creating databases

Exercise!

- Download and unpack **`python-libmount`**:
  - https://github.com/oucs/python-libmount/tarball/master
- Run **`dpkg-buildpackage`**
- Watch your terminal scroll like crazy
- Take a look inside **`debian/python-libmount/`**
- Notice the files in the directory above (e.g. **`python-libmount_0.9_all.deb`**)
- Use **`dpkg --contents <filename>`** to see what's inside the **`.deb`**

What just happened?

- **`dh_python2`** (remember that **`--with python2`**?) did the following:
  - noticed the **`setup.py`** in the root of the project
  - used it to install a copy in **`debian/python-libmount/usr/share/pyshared/`**
  - added links from **`/usr/lib/python2.X/dist-packages`** to **`…/pyshared`**
  - added something to the post-install script to byte-compile the Python files once they've been copied onto the target machine
- Aren't you glad you don't have to do this by hand?

Another exercise

- Let's package a really simple shell script
  - Download **`dev8d-cookie`** from https://github.com/alexsdutton/dev8d-cookie/tarball/master
  - Follow the instructions in the cunningly-named **`INSTRUCTIONS.txt`**

Repositories

- Briefly…
- **reprepro** is a tool for maintaining APT repositories
- Workflow:
  - Packages and their metadata dropped into an **incoming** directory
  - **reprepro** runs periodically from crontab, indexes them and removes them from **incoming**
- Add repository to **/etc/apt/sources.list**, after which an **apt-get update** will discover new packages, which can be installed as any other.
- http://www.debian-administration.org/articles/286 is a good guide.

reprepro repository layout

- conf/
  - distributions
    - ```
      Codename: mydistribution
      Architectures: source amd64 i386
      Components: main
      Description: Some text
      SignWith: <GPG KEY ID>
      ```
  - incoming
    - ```
      Name: default
      IncomingDir: incoming
      TempDir: tmp
      Allow: mydistribution otherdistribution>mydistribution
      Cleanup: on_deny on_error
      ```
  - uploaders
    - ```
      allow * by unsigned
      ```
- incoming/
- db/
- pool/
      ```

Repositories

- Ideas:
  - Get your CI server building packages and pushing them into a snapshot repo
  - Restrict access using Apache or SSH authorized_keys to keep your repo private
  - Combine with puppet or similar for nicely managed deployments
  - Dedicated account on repo box for managing repo; checks signature on incoming packages; uses passphrase-less GPG key for signing packages

Where now?

- The new maintainers' guide is imposing and huge, but useful
  - http://www.debian.org/doc/manuals/maint-guide/
- Debian has teams around similar types of packages
  - Join mailing lists, read team-specific documentation, ask questions
  - http://wiki.debian.org/Teams#Packaging_teams
- Look at pre-existing packages for clues on how to do things
  - http://anonscm.debian.org/viewvc/
- Realise there's so much to learn. Cry.

Contact

- alexander.dutton@oucs.ox.ac.uk
- alexsdutton on Twitter
- alex@jabber.ox.ac.uk if you do GTalk/Jabber/XMPP

- This talk is CC0. Do with it as you please.