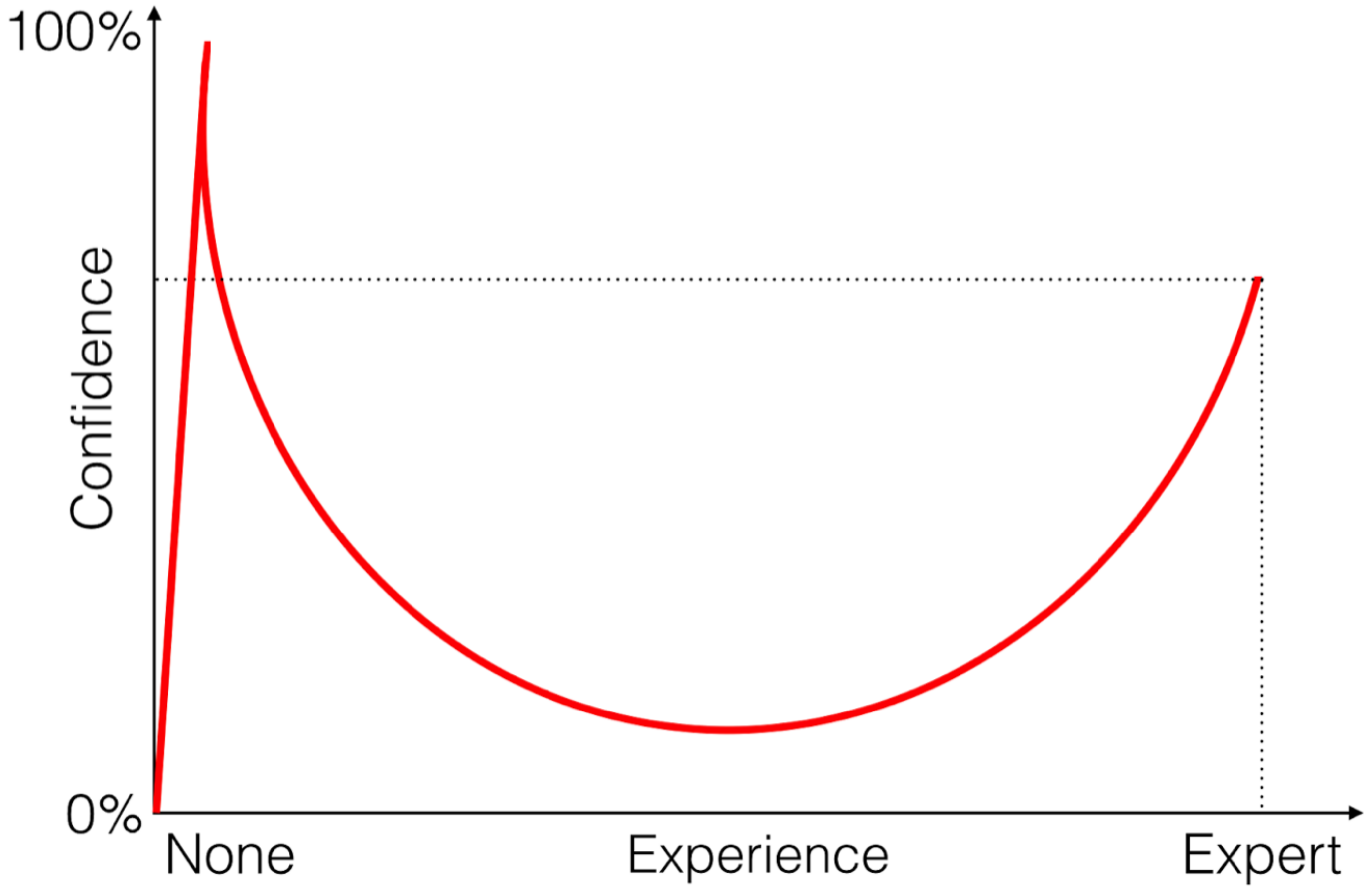
Making deb packages

Sweden c++ 20161215

Paul Dreik 2016
https://www.dreik.se/

Disclaimer



Dunning–Kruger effect

Today’s agenda

- Explain the basics
- Example 0: a shellscript
- Example 1: **Trompeloeil -** header only c++ lib
- Example 2: **nettle -** autoconf based c library
- Example 3: **SqliteCpp -** cmake based c++ library

Goal:

- You will be able to package your favourite software

What is a .deb package?

- Corresponds to a windows *.msi file
- Used by Debian, Ubuntu and derivatives
- Usually contains programs, documentation, libraries
- Can contain arbritrary data
- Has scripts to install and uninstall properly

Why package stuff?

- Makes it easy to install and deploy
- Easy to upgrade
- …and uninstall!
-                    Cleanly!
- Picks upp system wide flags
- Handles dependencies on other packages

It’s easy!

- Creating normal packages is easy
- The defaults are great
- The tools are very mature
- Lots of examples to study* : 20291 in Debian stable
- #apt-get source <package>

#on debian stable amd64, counting only official, free packages:

```
curl https://packages.debian.org/stable/allpackages?format=txt.gz | gunzip | tail -n+7
| cut -d' ' -f1 | xargs -n1 apt-cache showsrc | grep '^Package: '|sort|uniq |wc -l
```

It’s easy!

- Great documentation
- Tricky stuff is surprisingly easy
- daemons, creating users, pre-install scripts
- creating multiple packages from a single source etc

How it works

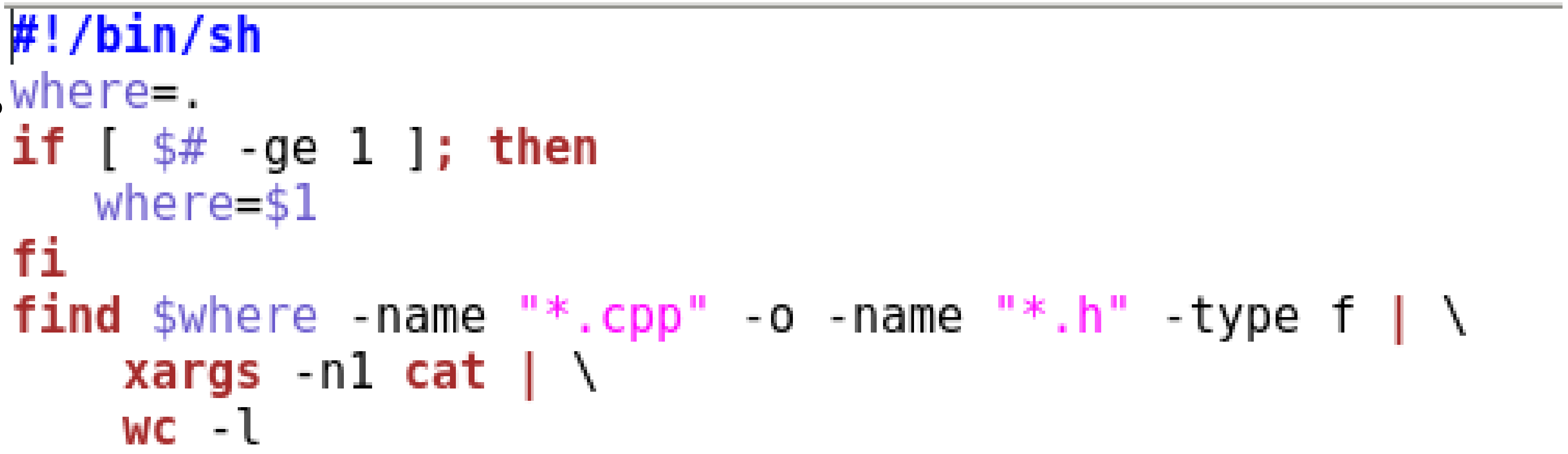
- Everything controlling the package is files in a debian/ dir
- Describe you package: debian/control
- (optional) tell how to build it: debian/rules
- (If necessary:) Tell what to install and where: debian/install

Getting started

1) Create initial packaging files
2) Add debian/* to version control
3) Manage the package version: dch -i
4) Build: dpkg-buildpackage -b -uc
5) Install: dpkg -i
6) Something wrong? Uninstall it with dpkg -r and repeat from 3!

Easy example – mission statement

- Let’s package a shell script
- Counts c++ source lines



```
#!/bin/sh
where=.
if [ $# -ge 1 ]; then
   where=$1
fi
find $where -name "*.cpp" -o -name "*.h" -type f | \
    xargs -n1 cat | \
    wc -l
```

- Mission: get this shell script properly installed

Easy example – first steps

- apt-get install dh-make build-essential
- (Demo time – switch to terminal)

// Here is what I will do in the terminal, for reference:

- dh_make --native --packagename countsourcelines_0.0 --single --yes
- Modify debian/control
- The package is now buildable!
- Build and install it
- dpkg-buildpackage -b -uc
- dpkg -i packagename_version.deb

Easy Example – make it useful

- Package does currently not do anything
- Modify debian/install
- count_source_lines usr/bin
- Pro tip: build it, then examine with tree debian/ to see what actually got included
- Done!
-
- ...but….

Easy Example – the ignored parts

- Package name
- Documentation – man page
- License
- Cooperation with other packages
- Dependencies
- Which of the points above is most important?

Easy Example - dependencies

- Our package needs find, sh, xargs, wc, cat
- Lets find out what to install.
- (Demo time)

- which find
- apt-file search bin/find |grep /find$
- Findutils, coreutils are needed

Easy Example – rev 2

Let us fix the missing dependency on findutils

- Edit debian/control
- Bump the version through debian/changelog: dch -i

Easy example – polishing

- Get rid of the example files in debian/*.ex
- Git ignore temporary build files
- Test build on multiple versions of debian, ubuntu, mint

Easy example - summary

We learned how to

- make the initial package files # dh_make
- how to build and install the package # dpkg-buildpackage -b -uc
- Install the package # dpkg -i
- how to work with dependencies # apt-file, which
- update the package # dch -i

Package Trompeloeil

- Let’s package Björn Fahller’s mocking framework!
- https://github.com/rollbear/trompeloeil
- Single header file: trompeloeil.hpp
- Goal: put it into /usr/include

- Why package it?
- Easy for other packages to depend on a minimum version!
- Build-Depends: trompeloeil(>=22)

Trompeloeil – first step

- apt-get install dh-make build-essential
- (Demo time – switch to terminal)

// Here is more or less what I will do in the terminal, for reference:

- git clone https://github.com/rollbear/trompeloeil.git
- cd trompeloeil
- git checkout v22
- git checkout -b debian
- dh_make --packagename trompeloeil_22 --native --single --yes
- emacs -nw debian/control
- dpkg-buildpackage -b -uc
- tree debian
- # nothing installed!

Trompeloeil – fix install

- Get trompeloeil.hpp into /usr/include
- Modify debian/install
- (Demo time – switch to terminal)

// Here is more or less what I will do in the terminal, for reference:

-
- emacs -nw debian/install
- dpkg-buildpackage -b -uc
- tree debian

Trompeloeil – fix license

- Fix license
- Modify debian/copyright
- (Demo time – switch to terminal)

// Here is more or less what I will do in the terminal, for reference:

- rm debian/copyright
- cd debian && ln -s ../LICENSE_1_0.txt copyright
- # repeat build etc., then
- dpkg -i trompeloeil*.deb

Trompeloeil - summary

We learned how to

- make the initial package files # dh_make
- how to build and install the package # dpkg-buildpackage -b -uc
- Inspect the package # tree debian
- Set the copyright # modify debian/copyright

Autoconf example

- Lets try to package nettle, a cryptography library in c
- https://www.lysator.liu.se/~nisse/nettle/
- Uses autoconf
- (Already available in libnettle-dev, so this is just for show)
- //demo time, switch to terminal

```
#Notes for the terminal session
wget https://ftp.gnu.org/gnu/nettle/nettle-3.3.tar.gz
tar xvzf nettle-3.3.tar.gz
cd nettle-3.3/
dh_make --packagename nettle_3.3 --native --single –yes
dpkg-buildpackage -b -uc
dpkg -i nettle*deb #as root
#done!
```

Cmake example

- Cmake projects are supported
- Easy if it provides an install target
-
- We will try to package a c++ library – sqlitec++
- https://github.com/SRombauts/SQLiteCpp

SqliteCpp – step 1

Create initial package

- // demo: switch to terminal

- dh_make --native --packagename sqlitecpp_2.0.0-1 --single –yes
- Build works, but nothing is installed!

SqliteCpp – step 2

- No cmake install target – complicates stuff.
- Where is the lib file?
- Where are the header files?
- Modify debian/rules #where to build
- Modify debian/install #what to install
-
- //demo time
- Use debian/rules from https://github.com/pauldreik/SQLiteCpp/blob/debpackage/debian/rules

SqliteCpp – summary

- Cmake made most of the work
- But needed tweaking!
- Unsolved questions: build with c++11? 14? default?

We are done! Recap time

How to package:

- Use dh_make
- Modify debian/control
- Modify debian/rules
- Modify debian/install
-
- Read the documentation, it is great!

Resources

- What the files in debian/ mean: https://www.debian.org/doc/manuals/maint-guide/dother.en.html
- How to package, in depth. Best Packaging Practices: https://www.debian.org/doc/manuals/developers-reference/ch06.en.html
- Rules on names etc: https://www.debian.org/doc/debian-policy/

Tools

- dpkg-dev-el # emacs highlightning
- apt-file # search package contents
- apt-get source # installs a source package
- dpkg-dev #for building
- dch # manipulating the changelog. Install devscripts

Bonus

- You can package data files
- You can set the compression level to speed up the build
- Reproducible builds