How to Update Ubuntu Packages

Setup and Configure a System to Build and Update Packages for Ubuntu

Prepared By: Steve Ovens

Initial Documentation: Jan 4, 2014

Applies to: Ubuntu 14.04

Table of Contents

Brief Overview

There are many different ways to update packages on Ubuntu this is simply one of the ways. In brief the steps are as follows:

1. Get the source package (apt-get source <package>)
2. Check the watch file, ensure that it has the desired regex (debian/watch)
3. Go into the source directory and run "uscan"
4. Run "uupdate ../<new package version>
5. In the new package directory, check the dependencies listed in configure.ac. Ensure these version numbers match what is found in debain/control.in
6. Once the control.in file is set, from the new package directory run

```
debuild -j6 -S
```

   (-j6 is similar to make flags, it determines the number of cores to use)
7. Once debuild has finished, attempt to create the package with pbuilder
8. If the package builds and installs correctly, create a debdiff to show the differences between the original package's DSC file and the newly created DSC

Back to Top

Using USCAN and UUPDATE

USCAN is a program which is reads the debain/watch file and then pulls down the source file from upstream. A sample watch file looks like this:

debian/watch

```
version=3
http://ftp.gnome.org/pub/gnome/sources/anjuta/([\d\.]+[02468])/ \
    anjuta-([\d\.]+)\.tar\.xz
```

Given the fact that development versions are given odd version numbers, the following regexes (in general) should be used:

```
stable_version = "([\d\.]+[02468])"
latest_version = "([\d\.]+)"
```

Once the watch file has the version regex that you need, simply run

```
builder@gnome-builder:~/build/anjuta/anjuta-3.8.4$ uscan
```

This will pull down the source code from the upstream.

The next step is to create the Debian/Ubuntu related files in the new upstream packages, to do this, use uupdate:

```
~/build/anjuta/anjuta-3.8.4$ uupdate ../anjuta_3.10.2.tar.xz
```

At this point you are ready to start examining the debuild steps

Back to Top

Setting up Debuild

There are a few things that should be considered while getting setup for using debuild. First you need to have already generated your own GPG key. In this example the GPG key exists and looks like this

```
/home/builder/.gnupg/pubring.gpg
--------------------------------
pub   4096R/C3A68603 2013-09-15
uid                  Steve Ovens <steve_ovens@linux.com>
sub   4096R/AC10BA50 2013-09-15
```

Once you have created your GPG key, you will need to add the following entries in your .bashrc file to assist the build process:

bashrc

```
export GPGKEY=C3A68603
export DEBEMAIL="steve_ovens@linux.com"
export DEBFULLNAME="Steve Ovens"
export DEB_BUILD_OPTIONS="parallel=6"
```

NOTE: the DEB_BUILD_OPTIONS are strictly optional and are intended to allow for the use of multiple cores when using debuild

After editing the bashrc file, you may wish to have the gpg-agent start when you log in so that the key ring will remember your gpg password for package creation.

.profile

```
gpg-agent --daemon --enable-ssh-support \
      --write-env-file "${HOME}/.gpg-agent-info"
if [ -f "${HOME}/.gpg-agent-info" ]; then
   . "${HOME}/.gpg-agent-info"
  export GPG_AGENT_INFO
  export SSH_AUTH_SOCK
  export SSH_AGENT_PID
fi
GPG_TTY=$(tty)
export GPG_TTY
```

Once this is completed, the next step is to review the dependencies of the package. To do so examine the <package>/configure.ac. In this case it would be anjuta-3.10.2/configure.ac. This file has all the requirements to build the upstream package.

configure.ac

```
GLIB_REQUIRED=2.34.0
GTK_REQUIRED=3.6.0
GTHREAD_REQUIRED=2.22.0
GDK_PIXBUF_REQUIRED=2.0.0
GDA4_REQUIRED=4.2.0
GDA5_REQUIRED=5.0.0
LIBXML_REQUIRED=2.4.23
GDL_REQUIRED=3.5.5
LIBWNCK_REQUIRED=2.12
GTKSOURCEVIEW_REQUIRED=3.0.0
VTE_REQUIRED=0.27.6
LIBDEVHELP_REQUIRED=3.7.4
GLADEUI_REQUIRED=3.12.0
GI_REQUIRED=0.9.5
NEON_REQUIRED=0.28.2
```

debuild looks in the debian/control.in file for any package dependencies.

debian/control.in

```
Build-Depends: debhelper (>= 8),
               cdbs (>= 0.4.90),
               dh-autoreconf,
               flex,
               bison,
               gnome-pkg-tools (>= 0.11),
               libglib2.0-dev (>= 2.34.0),
               libgtk-3-dev (>= 3.6.0),
               libgdk-pixbuf2.0-dev (>= 2.0.0),
               libgda-5.0-dev (>= 5.0.0),
               libvte-2.90-dev (>= 1:0.27.6),
               libxml2-dev (>= 2.4.23),
               libgdl-3-dev (>= 3.5.5),
               libgtksourceview-3.0-dev (>= 3.0.0),
               libdevhelp-dev (>= 3.7.4),
               libgladeui-dev (>= 3.12.0),
               libvala-0.20-dev,
               valac-0.20,
               gobject-introspection (>= 0.9.5),
               libgirepository1.0-dev (>= 0.10.7-1~),
               libneon27-gnutls-dev (>= 0.28.2),
               libsvn-dev (>= 1.5.0),
               pkg-config (>= 0.22),
               intltool (>= 0.40.1),
               gtk-doc-tools (>= 1.4),
               yelp-tools,
               autogen,
               python-dev
```

Once you have ensured that the dependencies are set in the control.in file, you need to update the debian/changelog. Please note the dependencies which were updated and any other change that was required.

After editing the changelog, you are ready to create your dsc file. This file will include all of the changes that have been made which are required to build the .deb package.

```
~/build/anjuta/anjuta-3.10.2 $ debuild -S -j6
```

This will create the DSC file, the "-j6" is similar to the make flag -j6, it is used to indicate the number of cores to use plus 1. (Therefore in this case it is set to use 5 cores). The DSC file is created at the top level of your directory tree. For example, given the directory structure:

```
anjuta/
 anjuta-3.10.2/
 anjuta_3.10.2-0ubuntu1.debian.tar.gz
 anjuta_3.10.2-0ubuntu1.dsc
 anjuta_3.10.2-0ubuntu1_source.build
 anjuta_3.10.2-0ubuntu1_source.changes
 anjuta-3.10.2.orig/
 anjuta_3.10.2.orig.tar.xz -> anjuta-3.10.2.tar.xz
 anjuta-3.10.2.tar.xz
 anjuta-3.8.4/
 anjuta_3.8.4-1.debian.tar.gz
 anjuta_3.8.4-1.dsc
 anjuta_3.8.4.orig.tar.xz
```

You can see the *.dsc files are in the root of "anjuta".

Back to Top

Setup Pbuilder

After you have dsc files, you are ready to setup PBUILDER, which will eventually create the DEB packages for you. There are a couple of files which you can edit, ~/.pbuilderrc and /etc/pbuilderrc. My preference is to edit /etc/pbuilderrc. This file needs to contain the list mirrors which you want PBUILDER to pull its build dependencies from. **ALLOWUNTRUSTED=yes** is a required field, particularly if you are running your own mirrors for testing. The **APTCACHEHARDLINK** is only required if you are planning to have PBUILDER work inside tmpfs

/etc/pbuilderrc

```
OTHERMIRROR="deb http://192.168.99.51/locally_built/ubuntu trusty main|deb
http://192.168.99.51/ca.archive.ubuntu.com/ubuntu trusty main restricted multiverse |
deb http://192.168.99.51/ca.archive.ubuntu.com/ubuntu trusty-updates main restricted
mulitverse"
ALLOWUNTRUSTED=yes
APTCACHEHARDLINK=no
COMPONENTS="main restricted multiverse universe"
```

By default, PBUILDER has its "scratch" directories located under /var/cache/pbuilder/build, in order to speed up the build process, this can be loaded into ram by adding the following to /etc/fstab:

/etc/fstab

```
tmpfs           /var/cache/pbuilder/build              tmpfs
defaults,size=6400M        0       0
```

The first time you work with pbuilder, you will need to tell it to create a clean environment for you. To do this run the following command:

```
sudo pbuilder --create --distribution trusty --debbuildopts "-I -i -j5" --compressprog
"pigz"
```

NOTE: –debuildopts and --compressprog are optional, and are used to enable multicore processing

This will take some time to setup the initial environment. After it is complete you are ready to make your first attempt at creating a package:

```
sudo pbuilder --build --debbuildopts "-I -i -j5" --compressprog "pigz" --distribution
trusty anjuta_3.10.2-0ubuntu1.dsc
```

If this completes successfully, you should find your new .deb in /var/cache/pbuilder/result along with several other files:

```
builder@gnome-builder:~/build$ ls /var/cache/pbuilder/result/*anjuta*
/var/cache/pbuilder/result/anjuta_3.10.2-0ubuntu1_amd64.changes
/var/cache/pbuilder/result/anjuta_3.10.2-0ubuntu1_amd64.deb
/var/cache/pbuilder/result/anjuta_3.10.2-0ubuntu1.debian.tar.gz
/var/cache/pbuilder/result/anjuta_3.10.2-0ubuntu1.dsc
/var/cache/pbuilder/result/anjuta_3.10.2-0ubuntu1~trusty1_amd64.changes
/var/cache/pbuilder/result/anjuta_3.10.2-0ubuntu1~trusty1_amd64.deb
/var/cache/pbuilder/result/anjuta_3.10.2-0ubuntu1~trusty1.debian.tar.gz
/var/cache/pbuilder/result/anjuta_3.10.2-0ubuntu1~trusty1.dsc
/var/cache/pbuilder/result/anjuta_3.10.2.orig.tar.xz
/var/cache/pbuilder/result/anjuta-common_3.10.2-0ubuntu1_all.deb
/var/cache/pbuilder/result/anjuta-common_3.10.2-0ubuntu1~trusty1_all.deb
/var/cache/pbuilder/result/anjuta-dbg_3.10.2-0ubuntu1_amd64.deb
/var/cache/pbuilder/result/anjuta-dbg_3.10.2-0ubuntu1~trusty1_amd64.deb
/var/cache/pbuilder/result/gir1.2-anjuta-3.0_3.10.2-0ubuntu1_amd64.deb
/var/cache/pbuilder/result/gir1.2-anjuta-3.0_3.10.2-0ubuntu1~trusty1_amd64.deb
/var/cache/pbuilder/result/libanjuta-3-0_3.10.2-0ubuntu1_amd64.deb
/var/cache/pbuilder/result/libanjuta-3-0_3.10.2-0ubuntu1~trusty1_amd64.deb
/var/cache/pbuilder/result/libanjuta-dev_3.10.2-0ubuntu1_amd64.deb
/var/cache/pbuilder/result/libanjuta-dev_3.10.2-0ubuntu1~trusty1_amd64.deb
```

At this point, PBUILDER has completed its job,

Back to Top

Creating the Debdiff

This is the simplist part of the process. Assuming that everything has gone well to this point you should have 2 .dsc files, the original which came down with the source package, and the one you created. In order to record the differences between package builds we run the following command:

```
~/build/anjuta $ debdiff anjuta_3.8.4-1.dsc anjuta_3.10.2-0ubuntu1.dsc |filterdiff -i
"*/debian/*" > anjuta_3.10.2.debdiff
```

Create a new bug on launchpad with the title "<update package> anjuta", and upload the debdiff as a patch.

After this, one of the more senior developers will review and upload the new package to the PPA

Back to Top