AugustinaBlair/PackagingGuideEdit

Introduction

What is Packaging?

Software applications in Ubuntu are distributed as packages via repositories. Package management tools in Ubuntu search the repositories for updates and new applications to install. Package managers include apt, Synaptic, and the Application Center. Package maintainers manage the software in the Ubuntu repositories. To learn more about installing applications on Ubuntu, see https://help.ubuntu.com /community/InstallingSoftware.

Packaging is the process of creating an archive of the files and directories an application needs for installation on an Ubuntu system. Building packages involves using tools that automate most of the process. This process includes compiling the package source and updating the information files that describe the application in package management tools. The complete package can then be submitted to the Ubuntu repository for review.

Who is this Guide For?

The Packaging Guide targets intermediate Ubuntu users who are comfortable with the Command Line Interface and have installed applications on Ubuntu.

The Packaging Guide is ideal for:

- Ubuntu users who want to learn more about how Ubuntu works.
- Ubuntu users who would like to contribute to the Ubuntu community.
- Ubuntu developers who would like to contribute their own applications or downstream patches to the Ubuntu repository.

Contents

1. Introduction
    1. What is Packaging?
    2. Who is this Guide For?
2. The Packaging Environment
    1. Overview
    2. Packaging Tools
        1. Build-Essential
        2. Devscripts
        3. Ubuntu-Dev-Tools
        4. Debhelper
        5. Dh_Make
        6. GnuPG
        7. Pbuilder
    3. Safe Packaging with Chroot and Pbuilder
3. Basic Packaging
    1. Overview
    2. Get Application Source Code
        1. Source Package Contents
        2. Source Code from a Tar File
        3. Apt-get Source
        4. Dget
        5. Extraction
    3. Edit the Source Files
        1. Run Dh_Make
        2. The Debian Directory
        3. Changelog
        4. Control
        5. Copyright
        6. Rules
        7. Compat
    4. Package the Application

The Packaging Environment

with Automation Tools
5. Submit the Application to the Ubuntu Repository
6. Basic Packaging Examples
    1. Example 1: Examine an Existing Package
    2. Example 2: Package GNU Hello From Scratch

Overview

The packaging environment in Ubuntu is a set of applications and scripts that streamline the packaging process.

Packaging Tools

To get started with packaging Ubuntu applications, install the following packages from the Ubuntu repository. Each is discussed in more detail in its own section below.

- build-essential
- devscripts
- debhelper
- dh_make
- gnupg
- pbuilder

The tools in this section are not required, but they do make the packaging process easier. Additional tools are discussed in the Advanced section. (Provide link once section is available)

Build-Essential

The build-essential package is a collection of standard development tools. Tools in other packages depend on these tools to create packages. For more information on the build-essential package, see the entry for build-essential in the Debian Packaging Manual - http://www.debian.org/doc/packaging-manuals/build-essential

Devscripts

The devscripts package contains scripts for automating packaging tasks. The primary tool used in this package is `debuild`. See the section Package the Application with Automation Tools to learn more about `debuild`.

Ubuntu-Dev-Tools

The ubuntu-dev-tools package is a collection of scripts specific for Ubuntu packagers.

Debhelper

The debhelper package contains scripts that perform common packaging tasks.

Dh_Make

dh_make creates the debian directory and generates files that describe the package.

GnuPG

All packages submitted to Ubuntu must be signed with a GnuPG, or GPG, signature.

To set up a GPG key, follow the instructions labeled "Using GnuPG to generate a key" here: https://help.ubuntu.com/community/GnuPrivacyGuardHowto

Pbuilder

Pbuilder is one of several tools available to test packages. Other tools, including sbuilder and the Personal Package Archives (PPA), are discussed in the "Advanced Packaging" reference.

Pbuilder uses chroot to construct a minimal Ubuntu installation for building and validating packages. Pbuilder was developed for package maintainers because `chroot` environments can be difficult to set up.

This section covers installing Pbuilder. See the section "Safe Packaging with Chroot and Pbuilder" for instructions for a minimal Pbuilder setup. See the PbuilderHowTo to learn more about Pbuilder: https://wiki.ubuntu.com/PbuilderHowto

Safe Packaging with Chroot and Pbuilder

Pbuilder uses Chroot to create an isolated Ubuntu environment with a minimal installation for building and validating packages. Chroot keeps the development environment separate from your desktop environment, preventing conflicts in your desktop environment from affecting packages in the development environment.

This section walks through a basic `pbuilder` setup. For more details, including how to target multiple versions of Ubuntu with `pbuilder`, see the PbuilderHowTo: https://wiki.ubuntu.com/PbuilderHowto

Create the chroot environment with pbuilder.

```
sudo pbuilder create -debootstrapopts -variant=build
```

`pbuilder` creates a base tar file in `/var/cache/pbuilder` that contains the `chroot` environment. The "`--variant=build`" option includes the build-essential package in the chroot environment which contains tools required for packaging.

Basic Packaging

Overview

The packaging process involves the following steps:

1. Get application source code
2. Edit files to describe the package.
3. Package the application with automation tools
4. Optionally, submit the package to the Ubuntu repository

Each of these steps is explained in detail in its corresponding section below.

See the tutorials under Basic Packaging Examples to apply the concepts described in each section.

Get Application Source Code

The source code for an application is required for the packaging process because the application needs to be built on the Ubuntu platform. Building on Ubuntu ensures compatibility and allows the package maintainer the opportunity to address any dependency issues.

The Basic Packaging section of the Packaging Guide covers three ways to get the application source code:

- Tar - For creating a new package that does not exist in the Ubuntu repository.
- Apt-get source - For working with a package that exists in the Ubuntu repository.
- Dget - For targeting a different version of Ubuntu than what is on your system.

Source Package Contents

A package contains the following file types:

- .dsc - Debian Source Control file containing information about the source package.
- .orig.tar.gz - The original source code, referred to as "upstream".
- .diff.gz - Changes, or patches, made to the original source code.

The .dsc file contains information that describes the package. The .dsc also contains md5 checksums that validate the integrity of the package. Md5 checksums are used by the installation process to ensure the archived source code isn’t corrupted. debuild, discussed in the section "Package the Application with Automation Tools", creates the .dsc file from a series of files dh_make generates during the packaging process.

Ubuntu runs its own version of applications, including Gnome and Debian packages. The upstream version of an application is the version provided by the original developers. The downstream version of an application is the version modified for use in Ubuntu. The .orig.tar.gz file is the original source code provided by the developer, or the upstream code.

The .diff.gz file contains a history of modifications that have been applied to the original

source code. When you download the source package from a repository, an extraction utility applies the .diff.gz file to the original source code extracted from .orig.tar.gz. apt and dget do this automatically on download. Applying the .diff.gz file creates the downstream version of the application Ubuntu users install from the repositories. See the Extraction section for alternate ways to apply the .diff.gz file.

Source Code from a Tar File

Download an existing tar file or create a tar file from files on your system. See Example 2 for one way to download a tar file.

If you are packaging software that is not available as a tar file, create the tar file from an unpacked source directory.

Make a directory labeled with the name of the application, a dash, and the version number of the application.

```
mkdir hello-2.4
```

Copy the files for the application into the newly created application directory.

```
cp myhello/* hello-2.4
```

Run Tar on the on directory to create the tar file.

```
tar czf hello-2.4.tar.gz hello-2.4
```

Tar creates a file labeled hello-2.4.tar.gz. Create a copy of the tar file with the “orig” extension to designate it as the .orig.tar.gz file in the package.

```
cp hello-2.4.tar.gz hello_2.4.orig.tar.gz
tar -xzvf hello_2.4.orig.tar.gz
```

Note the use of the underscore between the package name and the version number. This format is required by the Debian packaging tools. Without it, the tools do not recognize the original source and the package is built as a Debian native package.

Apt-get Source

Use `apt` to download source code that exists in the Ubuntu repositories. Modify the Software Sources list to include source repositories either by modifying the sources.list file or by changing the Repositories settings in Synaptic.

To modify the sources.list file, open `/etc/apt/sources.list` with root privileges.

```
sudo gedit /etc/apt/sources.list
```

Uncomment lines starting with "deb-src" by deleting the hash "#" character at the beginning of the line.

```
#  deb-src http://us.archive.ubuntu.com/ubuntu/ jaunty main restricted
```

For more information on using Synaptic to enable source downloads, visit https://help.ubuntu.com/community/InstallingSoftware.

Navigate to the directory where you want `apt` to download the source. From the terminal, run the apt-get source command with regular user privileges. NOTE: Most apt commands involve modifying files at the system level which requires root privileges. If you run apt-get source with root privileges, i.e. sudo, the resulting files will only be modifiable by a user with root privileges.

```
cd ~/hello
apt-get source hello-debhelper
```

`apt` downloads the source to the location where you run the command and extracts it.

Dget

`dget`, a tool in the devscripts package, downloads the source from a path you specify. Use `dget` to download packages from a different version than your system installation. For instance, if you want to build a package from a previous version of Ubuntu, use `dget` to download the package from the correct repository.

`dget` requires either the package name or a URL to a .dsc file. For more details on `dget`, from the command line, either type `dget --h` or type `man dget`.

Look up packages at http://packages.ubuntu.com for Ubuntu packages and http://packages.debian.org for Debian packages. Select the distribution version from the list shown. This returns a list of package sections. If you don't know which section your package is in, click on the link for all packages at the bottom of the page. This returns a list of all packages and can take a few seconds to load. Use Control-F to find your package. Click on its name to view the package information. The link to the .dsc file is on the far right column under "Ubuntu Resources". Click on the link to the .dsc file and copy the URL.

```
dget http://archive.ubuntu.com/ubuntu/pool/main/h/hello-debhelper/hello-
debhelper_2.2-3.dsc
```

Extraction

`dget` and `apt-get source` extract the package from the source files automatically on download. If you downloaded the individual archives through other means, unpack them using `dpkg-source`.

```
dpkg-source -x *.dsc
```

Edit the Source Files

Use `dh_make` to create the debian directory and the packaging information files. The debian directory separates the packaging information files from the application source files.

Run Dh_Make

From the source directory, run dh_make and specify an email address. This email address should be the same as the one the GPG certificate is assigned to. The packaging tools will verify this later in the build process.

```
cd hello-2.4
dh_make -e gpgemail@domain.com
```

Specify the type of package.

- Single Binary
- Multiple Binary
- Library
- Kernel Module
- CDBS

```
Type of package: single binary, multiple binary, library, kernel module or
cdbs?
[s/m/l/k/b]
```

dh_make fills out the rotation of the information based on system variables and values present in the source files.

```
Maintainer name  : Your Name
Email-Address    : gpgemail@domain.com
Date             : Thu, 6 Apr 2006 10:07:19 -0700
Package Name     : hello
Version          : 2.4
License          : blank
Using dpatch     : no
Using quilt      : no
Type of Package  : Single
Hit <enter> to confirm:
```

Press the enter key. `dh_make` confirms it created the files.

Only run dh_make once.

If you need to change any of the values you entered, remove the source directory and extract the tar file again. Repeat the steps to run `dh_make`.

```
cd ..
rm -r hello-2.4
tar -xzvf hello_2.4.orig.tar.gz
```

The Debian Directory

`dh_make` creates the `debian` directory containing the package information files and several template files ending in .ex. The basic packaging information files dh_make creates are:

- Changelog
- Control
- Copyright
- Rules
- Compat

Each file is discussed in the corresponding subsection below. See the Debian New Maintainer's Guide for more detail on each of these files: http://www.debian.org/doc/manuals/maint-guide/ch-dreq.en.html

Additional files `dh_make` adds that are not covered in this section:

- README.Debian: Debian and Ubuntu end-user documentation file.
- dirs: Used by dh_installdirs to create needed directories.
- docs: Used by dh_installdocs to install program documentation.
- info: Used by dh_installinfo to install the info file.

For more information about these files, see the section `dh_make` example files. Also see the Debian New Maintainers' Guide section that discusses these files in detail: http://www.debian.org/doc/manuals/maint-guide/ch-dother.en.html

Changelog

The Changelog file lists changes made in each version of the application. The specific format provides the package name, version, distribution, changes, and who made the changes. Use your GPG Key name and email address in the Changelog.

Look at the contents of the Changelog by using any text viewer. Here is a sample of what a new Changelog looks like:

```
cat changelog
hello (2.4-0ubuntu1) jaunty; urgency=low
   * Initial release (Closes: #1234)
--GPG Username <gpgemail@domain.com> Sun, 26 Jul 2009 10:44:53 -0700
```