Software Development on Linux Systems

4002-XXX-XX

By

Cody Van De Mark

Today

- Linux Distribution Differences
- Specialty Distributions
- Building RPMs and DEBS

Linux Distribution Differences

- GNU/Linux comes in many distributions
- This course deals with Fedora and Ubuntu
- The skills from Fedora and Ubuntu extend to other RedHat based distributions and other Debian based distributions
- It is important to note that there are other flavors of Linux that are entirely different

Linux Distribution Differences

- Main GNU/Linux Flavors:
  - Arch
  - Debian
  - RedHat
  - Gentoo
  - openSuse
  - Puppy
  - Slackware

Linux Distribution Differences

- Package Types:
  - Arch - pkg.tar.xz (TAR with LZMA2 compression)
  - Debian - DEB (Debian File)
  - RedHat - RPM (Redhat Package Manager)
  - Gentoo - ebuild
  - openSuse - RPM (Redhat Package Manager)
  - Puppy - pup
  - Slackware - tgz (TAR with LZMA2 compression)

Linux Distribution Differences

- Package Managers:
  - Arch - pacman (Package Manager)
  - Debian - apt (Advanced Packaging Tool)
  - RedHat - yum (Yellowdog Update, Modified)
  - Gentoo - portage
  - openSuse - YaST (Yet another Setup Tool)
  - Puppy - pupget
  - Slackware - installpkg

Linux Distribution Differences

- Each flavor has its own family of distributions
- Each distribution is designed for a specific change, purpose or audience
- Distributions may merge together or die off after their purpose has been served

Linux Distribution Differences

- General Flavor Purposes:
  - Arch - Minimalist distribution
  - Debian - Commitment to free software ideology using Debian architecture
  - RedHat - Commitment to free software ideology using RedHat architecture
  - Gentoo - Intended for power users, focusing on compiling code from source
  - openSuse - Novell sponsored Suse architecture distro
  - Puppy - Ultra lightweight and ease of use
  - Slackware - Super simplistic architecture design

Linux Distribution Differences

- Popular Debian Family Members:
  - Debian
  - Ubuntu family
  - Crunchbang
  - Knoppix

Linux Distribution Differences

- Popular Ubuntu Family Members:
  - Ubuntu / Kubuntu / Edubuntu / Lubuntu
  - Mythbuntu
  - Ubuntu Studio
  - Mint
  - GnewSense
  - EasyPeasy
  - BackTrack

Linux Distribution Differences

- Popular RedHat Family Members:
  - Red Hat
  - Fedora Family
  - Mandriva
  - PCLinuxOS
  - Oracle

Linux Distribution Differences

- Popular Fedora Family Members:
  - Fedora
  - Yellow Dog
  - CentOS
  - Scientific
  - Fermi

Linux Distribution Differences

- Common Desktop Environments:
  - Unity
  - Gnome 2
  - Gnome Shell
  - LXDE (Lightweight X11 Desktop Environment)
  - Xfce (acronym no longer stands for anything)
  - Sugar (used for One Laptop Per Child)

Specialty Distributions

- There are many specialty distributions of Linux that are designed for very specific purposes
- Many of these are not able to be installed and are only live versions
- Others are designed only for embedded systems and will more as firmware than an actual operating system

Specialty Distributions

- Common Specialty Distributions:
  - Backtrack - Penetration Testing
  - Yellow Dog Linux - Designed for PowerPC
  - OpenWRT - Embedded router software
  - XBMC Live - Xbox Media Center Live CD
  - SuperGamer - Live gaming distribution
  - Puppy - Lightweight, can run in RAM
  - Damn Small Linux - Tiny 50mb distro for old hardware
  - Slitaz - Tiny 30mb distro for small hardware
  - GParted Live - Bootable partition editor/recovery
  - Ubuntu Studio - Multimedia production distro

Linux Distribution Differences

- As you can see, Linux has a very wide variety of distributions and flavors
- Special distributions exist for many purposes, some more specific than others
- This course is specific to Fedora and Ubuntu as they are two of the most popular distributions and have large families
- It is likely you will work with Debian or RedHat systems the most, but you are encouraged to look for the tool you need

Building RPMS and DEBS

- Now that you have seen the package families, we will look at RPMs and DEBs specifically
- DEB files and RPM files are the two most common binary package types on Linux
- DEBs and RPMs are installer packages built from source code
- It is important to note that with a little work, RPMs can be used on Ubuntu and DEBs on Fedora

Building RPMS and DEBS

- DEBs and RPMs are essentially the equivalent if EXE files on Windows or DMG files on Mac OS X
- DEBs and RPMs are very detailed unlike EXE or DMG files
- DEBs and RPMs hold management information, such as version, dependencies, architecture types, licensing, suggested add-ons, etc
- DEBs and RPMs also hold enough information to resolve conflicts
- You are required to hand in a DEB or RPM file for your project

Building RPMS and DEBs

- You will need to build an RPM and DEB in lab
- This requires
  - Creating a GPG signature key
  - Creating a source code package with Make files
  - Creating DEB configuration files
  - Creating RPM configuration files

GPG Keys

- GPG is Gnu Privacy Guard and is an alternative to Pretty Good privacy (PGP)

- GPG keys are public/private pair encryption keys;

  Files signed with the private key identify you are the creator as no one else can create or use your key

- GPG are essential to preventing fraudulent and malicious packages into Linux;

  Your system will not want to accept packages from unknown sources unless your force it to

Source Packages

- Source packages are a package comprised of a source code folder that has been compressed into tar.gz format
- Source packages include all of the files a user needs to compile, make, install and use software
- This includes:
  - Source code
  - System configuration files
  - Make files
  - Documentation (Licensing, man pages, README, etc)

Source Packages

- The special components of a source package are the configuration file and the make files which make the source code portable;

  You will almost always find these in the source tar.gz

- These are what make packages and installers possible;

  Both are needed for an RPM, DEB file or other installer packages

Source Packages

- The **configure** file is a script that determines all of the specifics about your current machine, such as
  - Cpu Architecture
  - Graphics Card
  - Compilers
  - Operating System
  - Kernel
  - etc

- End users who download source code, can run the configure script and have it modify the make files to work on that specific hardware

Source Packages

- The **make** file is a script produced by the **configure** script that compiles all of the source files and makes a binary file based on the specific requirements of that machine

- After the make script has been run, users will be left with an executable file to run and use your software

- After the make script has been run, users may run **make install** which will take the new executable file and install it to the machine with the documents provided

Source Packages

- It is not uncommon for people on Linux to directly download source code and install it themselves this way
- The disadvantage to this is the process is more difficult and it is unlikely you will receive software updates as the package did not go through the package manager
- This processes is referred to as compiling from source and is done with three commands
  - ./configure
  - make
  - make install

Source Packages

- If you have produced the configure and make files to allow this process, you can zip the entire folder up into a tar.gz folder
- This is referred to a source package or “tarball”
- This can be distributed as is (and often is), but once this is made you can make RPM and DEB files to allow:
  - one click installs
  - package management
  - updates

DEB Files

- If you have an existing source package you can “Debianize” it with the command **dh_make**

- dh_make takes a source package file and adds a Debian directory full of DEB configuration files based off of your source package

- In the Debian directory, there are several important files
  - control - Build and Management Information
  - changelog - Changes listed by version
  - copyright - Copyright information and licenses
  - README.Debian - Project README file

DEB Files

- Control file sections:
  - **Source - Source package you are building from**
  - **Section - Section for project to be categorized in**
  - **Priority - Importance level of this package**
  - **Maintainer - Maintainer/team name and email**
  - **Build-Depends - Requirements for compiling**
  - **Standards-Version - Which Debian file version you want to use**
  - **Homepage - Project homepage**
  - **Package - Project name**
  - **Architecture - Which architecture(s) are supported**
  - **Depends - Requirements for installations/usage**
  - **Description - Project description**

DEB Files

- Example Control File *Note this is very sensitive to formatting

```
Source: hellotest
Section: devel
Priority: extra
Maintainer: Dr Professor <drProf@rit.edu>
Build-Depends: debhelper (>= 7.0.50~), autotools-dev
Standards-Version: 3.9.1
homepage: http://ist.rit.edu

Package: hellotest
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: This will print hello world this is my one space
 indented longer description. It must be indented one space
 in from the word description or else it will not parse correctly.
```

DEB Files

- Example Control File with Spaces

```
Source: hellotest
Section: devel
Priority: extra
Maintainer: Dr Professor <drProf@rit.edu>
Build-Depends: debhelper (>= 7.0.50~), autotools-dev
Standards-Version: 3.9.1
homepage: http://ist.rit.edu

Package: hellotest
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: This will print hello world this is my one space
 indented longer description. It must be indented one space
 in from the word description or else it will not parse correctly.
```

DEB Files

- You may find the dependencies for the project by running **dpkg-depcheck -d ./configure** against your configure file

- The changelog, copyright and README are the same as the Debian formats we discussed the documentation lecture

- In fact, you will replace the file README.Debian with your README file;

  Your replacement file should be named README, not README.Debian

DEB Files

- After you have filled out all of the required files you can build the DEB package with **dpkg-buildpackage -rfakeroot**

- This will produce a usable Deb file that can be installed with the **dpkg** command

- Debian is very strict for quality assurance reasons and most package managers and software centers will not yet accept this package;

  It will likely give a “Bad Quality” error

DEB Files

- There is a special package error tool called **lintian** which you can run against your package to find errors
- This tool will give you a thorough list of all of the errors in your DEB file
- Once these are corrected corrected and the DEB file is rebuilt, the package will be accepted by package managers/software centers

DEB Files

- Tips for building DEB Files in lab
  - Uncompress the source tar and make two copies of the folder
  - Also, make two copies of the original source tar
    This is done in case you mess up the first one, you can restore it

    Put the second and the second folder and the second source tar into a folder called Before Debianization

  - After you Debianize the file and fill out the correct information, make a copy of the entire source folder and the source tar

    Again, this is done in case you mess up the first one

    Put these into a folder called Before Build

RPM Files

- If you have an existing source tar, you can start the RPM process with **rpmdev-setuptree**
- This will create a folder called rpmbuild in your home folder
- Inside of rpmbuild is a folder called SOURCES, you will make a copy of the source tar in that folder still compressed as tar.gz
- Inside of rpmbuild is a folder called SPECS, in this you can run **rpmdev-newspec projectname** to create a new specifications file

RPM Files

- Spec file sections:

Name	**-**	**Project Name**
Version	**-**	**Version**
Release	**-**	**Release Number (aka Update number)**
Summary	**-**	**Brief overview of project**
License	**-**	**License type**
URL	**-**	**Homepage**
Source0	**-**	**URL of package itself for download**
BuildRequires	**-**	**Compiling dependencies**
Requires	**-**	**Installation and usage dependencies**
%description	**-**	**Project description**
%prep	**-**	**Preparation scripts**
%build	**-**	**Building scripts/flags**
%install	**-**	**Installation scripts/flags**
%files	**-**	**Files to install including binary and documents**
%changelog	**-**	**Changes listed by version**

RPM Files

- Example Spec File (Part 1):

```
Name:          hellotest
Version:       1.1
Release:       1%{?dist}
Summary:       This is the standard hello world program
License:       GPLv2+
URL:           http://ist.rit.edu
Source0:       http://isr.rit.edu/packages/hellotest-1.1.tar.gz

#BuildRequires:
#Requires:

%description
Hellotest is a program to print hello world and illustrate how RPMS work

%prep
%setup -q
```

RPM Files

- Example Spec File (Part 2):

```
%build
%configure
make %{?_smp_mflags}

%install
rm -rf $RPM_BUILD_ROOT
make install DESTDIR=$RPM_BUILD_ROOT

%files
%doc %{_mandir}/man1/hellotest.1.gz
"/usr/sbin/hellotest"

%changelog
* Thu Jan 24 2012 Dr Professor <drProf@rit.edu> 1.1-1
- This is the initial release
```

RPM Files

- You may now build the initial files with
  rpmbuild -ba projectname.spec

- After the initial files have been made and the spec file has been filed out appropriately, you can build the rpm with
  rpmbuild -ba projectname.spec –clean

- This will produce a usable RPM file that can be installed with the **rpm** command

RPM Files

- Similarly to Debian, most package managers and software centers will not yet accept this file for quality assurance reasons

- RPMS are also strict on package rules and offer a special tool to run against the RPM to find errors;

  This is called **rpmlint**

- This will give you a thorough list of errors in your RPM package

- Once you correct these errors and rebuild the package, it will be accepted by package managers/software centers

RPM Files

- Tips for building RPM Files in lab
  - Create the rpmbuild directory and put a copy of the source package inside of the SOURCES folder
  - Make a copy of the rpmbuild directory
    This is done in case you mess up the first one, you can restore it

    Put the second folder into a folder called Before Initialized Files

  - After you create the spec file, fill out the correct information and do the initial rpmbuild, make a copy of the rpmbuild directory

    Again, this is done in case you mess up the first one

    Put this into a folder called Before Build