R • I • T	Rochester Institute of Technology Golisano College of Computing and Information Sciences Department of Information Sciences and Technologies

4002-XXX
Software Development on Linux Systems
Lab 08 – Building and installing RPMs and DEBs

Name: ______________________________________ Section: _____________

Overview

In this lab you will be building DEB files and RPM files. These are basically the Linux equivalent of EXE files on Windows or DMG files on Mac OS X. DEB files and RPM files, however, are fairly detailed and hold a lot of information about the software including its version number, dependencies, architecture, license, etc.

You will need to use both Ubuntu and Fedora for this lab. The standard package for Ubuntu is a deb file, while the standard package for Fedora is an RPM. Though it is possible to use RPMs on Ubuntu or DEBs on Fedora with little work, we will not be covering that in this lab. It is good to note, that most distributions can handle packages from most other distributions with ease. One of the packages that makes this possible is Alien, a package conversion tool.

Note: To build packages for Java or Python for your project, you may want to use checkinstall for DEB files instead of dh_make

Activity – Installing Software Tools

Using your preferred method of installation install the package building tools
These are the package names-

Ubuntu: autoconf
automake
autotools-dev
dh-make
debhelper
devscripts
fakeroot
xutils
lintian
pbuilder

Fedora: @development-tools
fedora-packager

Activity – Create a GPG

In order to sign packages to verify where they came from, you need to create a signing key. You will create a GPG key to sign your packages with.

1) In terminal type in

   gpg --gen-key

2) Type **1** for RSA and RSA (default) then hit enter

3) Enter **2048** for encryption and hit enter

4) Enter **3m** for expiration in 3 months and hit enter

5) Enter **y** for yes and hit enter

6) Enter your full name (or alias) and hit enter.

   NOTE: We will use this name later in the lab

7) Enter your email address and hit enter.

   NOTE: Again we will use this later in the lab

8) You may leave the comment blank and hit enter

9) When asked

   Change (N)ame, (C)omment, (E)mail, or (O)kay/(Q)uit?

   Choose **o** and hit enter

10) Enter a password and confirm the password

   NOTE: We will use this later in the lab

   (You may need to type a lot into gedit or something to generate entropy for a truly unique encryption key)

11) Now you need to generate a public key

    Type
    gpg -a --output ~/.gnupg/*YOURNAME*.gpg --export '*YOUR NAME*'

    Where **yourname** is the name you want to give your key (your last name recommended)

    and **'YOUR NAME'** is the name you used when generating your key.

    This will create a public key based on the key you just name.

12) Now import that key into the current system so your system will accept packages signed by you as valid packages.

    Type

    gpg --import ~/.gnupg/*YOURNAME*.gpg

    where YOURNAME is the name you gave your public key in the previous step.
    (This will be your last name if you followed the recommendation).

Activity – Make files for packages

Before we make a DEB package or RPM we need to compile and create MAKE files for all of the code. Make files allow software to be configured and compiled for a specific system. As many end users have various hardware and system settings, it is important to make the software correctly.

1) Download the files **hello.cpp, hello.h** and **README** from mycourses

2) Create a folder called **hellotest-1.1**

3) Inside of this folder, create folders called **src** **doc** and **man**

4) Put the **hello.cpp** file and the **hello.h** file into the **src** directory

5) Put the **hellotest.1** file from the previous lab in the **man** folder

6) Put the **README** file you downloaded in the **doc** folder.

7) In the **hellotest-1.1** directory, run the command **autoscan**
   This will scan for your code and create a configuration file based on it

8) Rename the file **configure.scan** to **configure.ac**

9) Open the **configure.ac** file and change the line

   AC_INIT([FULL-PACKAGE-NAME], [VERSION], [BUG-REPORT-ADDRESS])

   to

   AC_INIT([hellotest], [1.1], [your@email.address])

10) Save the **configure.ac** file

11) In the **hellotest-1.1** directory run the command **autoconf**
    This will create a configuration file for end user systems and a folder for compiling tools

12) Now we need to configure the make file for compiling code. In the **hellotest-1.1** directory create a file called **Makefile.am (This is case sensitive).**

13) Open the new **Makefile.am.** We will need to add the foreign flag to indicate that we are not using the GNU format. We also need to list the directories we will be using.

add the following lines and save the file.

```
AUTOMAKE_OPTIONS = foreign
SUBDIRS = src doc man
```

14) Now you need to create a Makefile.am for the src folder.

In the **src** folder create a file also called **Makefile.am**. This file will contain flags for our source code, the name of the executable and the source files.

Add the following lines and save the file

```
CFLAGS =
LDFLAGS =

#This tells the compiler that this will be installed in sbin with the name hellotest
sbin_PROGRAMS = hellotest
hellotest_SOURCES = hello.cpp hello.h
```

15) Now in the **man** folder create a file called **Makefile.am**. This file will contain a list of which man files you will be installing.

Add the following line to the file and save it

```
man_MANS = hellotest.1
```

16) In the **doc** folder create a file called **Makefile.am**. This file will contain information about which directory to package documents and which documents to package.

Add the following lines and save the file.

```
docdir = $(datadir)/doc/@PACKAGE@
doc_DATA = README
```

17) In the **hellotest-1.1** folder, open the file **configure.ac** again.

    Add the line
    AM_INIT_AUTOMAKE(hellotest, 1.1)

    after the line
    AC_CONFIG_HEADERS([config.h])

    then replace the line

    AC_OUPUT

    with

    AC_OUTPUT(Makefile src/Makefile doc/Makefile man/Makefile)

    and save the file

18) In the **hellotest-1.1** folder run the command **autoheader** to create a template header file for the configuration file to use

19) Now run the command **aclocal** to automatically prepare the files for configuration

20) Run the command **automake - -add-missing** to automatically compile all of the files and add any default or missing scripts. It may tell you not to override from variables; Ignore that.

21) Now run the command **autoconf** again to modify the configuration script for end users.

    This should leave you with a file called **configure** that can be used to prepare make files for various hardware, operating systems and architectures (32 bit, 64 bit, etc).

    This is the state in which the code can be given to end users to compile. Open source code downloads are usually in tar.gz files that contain the directory at this state.

22) Run the **configure** file with **./configure**

    This will leave you with a valid **make** file that can be used to automatically compile all of your code.

23) Now in the **hellotest-1.1** directory run the command **make**

    This will automatically compile all of your code for the current system. If there are any errors it is likely due to a typo in one of the previous files.

24) Now if you go into the **src** directory you should see an executable called **hellotest**. You should be able to run this. We will now proceed to making the installers for this code.

25) Compress the **hellotest-1.1** folder to a tar.gz called **hellotest-1.1.tar.gz**

Instructor/TA Sign-Off ______________________________

Activity – Creating a DEB File

1) Transfer the **hellotest-1.1.tar.gz** file to your Ubuntu machine if you have not already.

2) Uncompress the compressed file, but keep the compressed file as well.

    At this point you should have a folder called **hellotest-1.1**

    and a compressed file called **hellotest-1.1.tar.gz**

3) cd to the **hellotest-1.1** directory and run the command

    dh_make -e *your@email.address* -f ../hellotest-1.1.tar.gz

4) When asked to choose a binary type

    type **s** and hit enter to choose single binary.

    This will create a debian folder in the hellotest-1.1 directory that contains files needed to create the DEB file.

5) cd to the new **debian** folder inside of the **hellotest-1.1** folder

6) Open the file named **control**

   Replace the line

   homepage: <insert the upstream URL, if relevant>

   with

   homepage: http://ist.rit.edu

7) Replace the line

   Section: Unknown

   with

   Section: devel

   This will tell the package which section to be listed under when people are searching for packages.

8) Replace the lines

```
Description: <insert up to 60 chars description>
 <insert long description, indented with spaces>
```

   with

```
#EVERY LINE AFTER THE ONE SAYING DESCRIPTION MUST BE
#INDENTED WITH A SPACE
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.
```

9) On the line

   Maintainer: ***lastname <your@email.address>***

   replace lastname with your lastname. This should be the same one you used for your gpg key. The email should already be the same, but if not, correct that as well.

10) Save the file. It should now look something like this. You may remove the vcs comments if you have them.

```
Source: hello test
Section: devel
Priority: extra
Maintainer: lastname <your@email.address>
Build-Depends: debhelper (>= 7.0.50~), autotools-dev
Standards-Version: 3.9.1
Homepage: http://ist.rit.edu

Package: hellotest
Architecture: all
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: This is print hello world
 This is my description of the hello world program
 It is quite a bit longer
```

11) Now in the **hellotest-1.1** folder run the command

```
dpkg-depcheck  -d  ./configure
```

   This will list any packages that are missing that your DEB file requires. They will be listed under **packages needed.** This is critical both to this and to your project. It is possible you may not be missing any files in this example, but you probably will be with your project.

12) Open the **control** file again in the **debian** folder

    Add all dependencies (if there were any) from the previous step to the control file with comma separation to the following line.

    Build-Depends: debhelper (>=7), autotools-dev

    Example:

    If dpkg-depcheck listed libc6-i686 and gawk your build-depends line would like this

    Build-Depends: debhelper (>=7), autotools-dev, libc6-i686, gawk

    Note: You must get this right for the control file to work, otherwise the DEB file will not work.

13) Save the control file

14) Download the **copyright** file from mycourses and replace the one in the **debian** folder with the one you downloaded.

    This will overwrite the default copyright file with one that has already been formatted. You may use this as reference for your project. Spacing is extremely critical in this file.

15) Open the copyright file and replace **EVERY** occurrence of programmerGuy@guy.it.us.edu with the email listed in the control file. Change **EVERY** occurrence of **programmerGuy** to the name you used in the control file.

    Save the copyright file

16) Replace the **changelog** file in the **debian** folder with the one you created for hellotest in the previous lab or modify the current one to look like the one you did in the previous lab. Spacing is extremely critical in this file. Make sure the version number does not have a -1 or anything after it. This will cause problems.

17) Now open the **changelog** file and replace all of the programmer **names/usernames** and **emails** with the **name** and **email** you used when you generated your key earlier in the lab. This is critical. All maintainer names and emails must match the ones you gave in the gpg key.

Remember that this file is extremely sensitive to spacing.

Also, replace
*** Initial release (Closes: #nnnn) <nnnn is the bug number of your ITP>**

with

*** Initial release**

BE SURE TO GET THE SPACING EXACTLY RIGHT. THERE SHOULD BE TWO SPACES BEFORE THE * AND ONE AFTER

18) In the **debian** folder replace the **README.Debian** with the **README** file you put into the **doc** directory.

Make a copy of that file in the **debian** directory and remove both the **README.Debian** file and the **README.source** file.

You should only have the **README** file left in the **Debian** folder

19) Now in the **hellotest-1.1** folder make the deb file with

```
dpkg-buildpackage  -rfakeroot
```

When prompted for the GPG key password, type in the password you gave when you generated in key.

If this gives you any errors it is likely due to spacing, incorrect name/email or typos in the changelog, copyright or control file.

You can now install this file with
If you used 64 bit:

```
sudo  dpkg  -i  hellotest_1.1-1_amd64.deb
```

If you used 32 bit:

```
sudo  dpkg  -i  hellotest_1.1-1_i386.deb
```

Note: The filename MIGHT be hellotest_1.1 instead of hellotest_1.1-1

To test: After installation open up terminal and run the command **hellotest.** If it prints "hello world" the installation worked.

Bonus) Now in the directory with your new **hellotest-1.1** deb file

run the following command
If you used 64 bit:

```
lintian  hellotest_1.1-1_amd64.deb
```

If you used 32 bit:

```
lintian hellotest_1.1-1_i386.deb
```

Note: The filename MIGHT be hellotest_1.1 instead of hellotest_1.1-1

Correct any errors that were reported. Chances are there were quite a few that are usually easily fixed with spacing, etc.

Redo step 18 and the bonus again to see if you got all the errors. If lintian does not report any errors, then your package passes all integrity checks and will be accepted by software managers, such as Ubuntu software center, synaptic or aptitude.

Activity – Creating an RPM File

1) Transfer the **hellotest-1.1.tar.gz** file to your Fedora machine if you have not already.

2) Now create an rpmbuild area with the following command

   rpmdev-setuptree

   This will create a folder in your home folder called **rpmbuild**

3) In the new **rpmbuild** folder there will be a folder called **SOURCES**

   Move the **hellotest-1.1.tar.gz** file into the **~/rpmbuild/SOURCES/** directory

4) Move to the **rpmbuild** folder called **SPECS** and issue this command to create the RPM specifications.

   rpmdev-newspec hellotest

5) In the **SPECS** folder open the new file **hellotest.spec**

   Enter the information for the following fields and ensure you match spacing/tabs

   For the **version** put in **1.1**

   For the **summary** put in **This is the standard hello world program**

   For the **license** put in **GPLv2+**

   For **URL** put in **http://ist.rit.edu**
   This is the page that the package will list as the official website. It is not actually used by the package.

   For **source0** put in **http://ist.rit.edu/packages/hellotest-1.1.tar.gz**
   This is the page that the package will list as a download for the source code tar.gz file. It is not actually used by the package.

6) For **%description** enter a description **on the line AFTER**

   For example:

   %description
   This is the "hello world" program. This is a test package built to show how RPMs work. It contains c++ source code and is installed as a command line utility.

7) For **%changelog** enter the date, your name, your email and the version number in the format from lab 7. Then put in "- This is an initial release"

   For example:

   %changelog
   * Thu Jan 24 2012 *lastname* <*email@rit.edu*> 1.1-1
   - This is the initial release

8) Put # in front of the lines **BuildRequires:** and **Requires:**

   Example:
   #BuildRequires:
   #Requires:

9) **Save** the **hellotest.spec** file

10) Now you need to build the initial RPM files

    To do this issue the following command in the **SPECS** folder

    rpmbuild -ba hellotest.spec

    Note: It may give you errors about unpackaged files. You can ignore these.

11) Open the **hellotest.spec** file again

    Under **%files** add the following lines with the quotes
    "/usr/sbin/hellotest"
    "/usr/share/doc/hellotest /README"

    Note: Notice the space between hellotest and /README. You may not need this. It may or may not give you an error depending on the version. If may give you a file not found error. If so either remove the space or look at the error and see if there are any typos.

12) Now you need to add the README location to the spec file.

    Change the line **%doc**
    to

    %doc %{_mandir}/man1/hellotest.1.gz

13) **Save the file**

14) In the **SPECS** directory issue the following command to build the new RPM file

    rpmbuild -ba hellotest.spec - -clean

    Note: If you get a file not found error look at the path and find the offending line in the hellotest.spec file. If it is the one with the space, remove the space and try again. Also, ensure the file exists at the location referenced in the error message. Make sure the path in the spec file matches that path. Redo this step to rebuild the RPM file. If there are any other errors, correct them before moving on.

15) In the **rpmbuild** folder go into the directory **RPMS** and the directory under **RPMS.** You should see RPM files. One will be the RPM and another will be a debugging version.

    You may test the installation by double clicking on the package and installing

    or by using the following command as root

    " rpm -i hellotest-1.1-1*.rpm "

Bonus) In the **SPECS** folder you may run the following command to test your new package for integrity. This is similar to the Bonus section of the DEB activity using lintian.

Run the following command:

```
rpmlint  hellotest.spec  ../SRPMS/hellotest*  ../RPMS/*/hellotest*
```

This may give you a list of errors. Correct any errors that were reported and redo step 15. Then you may use the rpmlint command again. If the command does not list any errors, then your package has passed all of the integrity checks and can be accepted by most RPM software managers.

You may ignore any url errors as the urls we have provided are not real