Lab 04: Package management

Hands-on Unix system administration DeCal

2012-02-13, due 2012-02-27

Building a Debian package

Debian has a robust binary package management system, and in almost all cases, you'll never need to compile or build software from source.

But as a Unix sysadmin, you might need more flexibility:

- use a newer version of package which is unavailable in the repositories (hasn't yet been packaged)
- apply a patch for further customization or bugfixes
- contribute to Debian by packaging

In this lab, you will (re)build a Debian package from source. It should work on any Debian-based system, including most OCF machines, which you should have an account on. First you will fetch the source package, which contains the software source code and some metadata, and then compile it into a binary package which can be installed with `dpkg`.

1. First, check that some packages useful for building Debian packages are installed. They are `build-essential` and `devscripts`. You can check if a package is installed with `dpkg -l foobar`, and install it with `aptitude install foobar`. You need root privileges to install packages since they get installed systemwide. *Explain why you might not have root privileges on an OCF computer.*

2. Fetch the source code of `hello` (as in "hello world"). Change to a temporary directory, such as `/tmp`, and make a directory inside it using `mkdir` and change to that directory. From the directory, you can run `apt-get source hello` to fetch the source package. *Describe how `apt-get source` works.* Hint: `man apt-get` to find out what `apt-get source` does, in particular the role of the `source.list` file (take a look at `/etc/apt/sources.list`).

3. In your current directory, there should be a directory, a `*.debian.tar.gz` file, a `*.dsc` file, and a `*.orig.tar.gz` file. The `*.orig.tar.gz` file contains the "upstream" source code for the program, in a compressed tarball. The `*.debian.tar.gz` file (or in most other cases, `*.diff.tar.gz`) contains Debian-specific patches including packaging information. The `hello*` directory contains the source code, with the Debian patchset applied. *What do you think is the purpose of the `*.dsc` file?*

4. Change your working directory to the `hello*` directory. There should be a directory called `debian`. This directory is the only difference between a Debian source package and a package meant to be `make install`ed. Change your working directory to the `debian` directory. *Describe what you think are the functions of the `changelog` and `control` files.*

5. Now move up one directory. You should again see the `debian` directory, and some other files, including a `Makefile` and `configure` script, as is standard for GNU packages. Now let's build the binary Debian package! Run the command `debuild -us -uc` (the extra arguments tell `debuild` not to sign the package, because we are not the package maintainer). *Which package provides debuild?* Hint: use `dpkg -S`. In case you're wondering, the `debian/rules` file takes care of calling `./configure` and `make` with the appropriate options when you build the source package, which is what you're seeing in the terminal.

6. Move up again another directory. You should now see a `*.deb` file. This is the Debian binary package you built, which can be installed with `dpkg -i` (you need root privileges, see question 1). Extract the file with the command `ar vx *.deb`. Three files should have been extracted, `debian-binary`, `control.tar.gz`, and `data.tar.gz`. Now extract these two files with `tar xzvf control.tar.gz` and `tar xzvf data.tar.gz`. *What are the dependencies of this binary package?* Hint: take a look at the `control` file which was extracted from `control.tar.gz`. *Describe the contents of this binary package.* Hint: take a look at the files extracted from `data.tar.gz`. You can also lookup information on the binary package by using `dpkg-deb` (run `dpkg-deb --help` for info).

Finally, clean up by removing the files you've created.