Guidelines for deploying preservation tools and environments

Authors
Rainer Schmidt (AIT Austrian Institute of Technology), David Tarrant (Open Planets Foundation), Rui Castro, Miguel Ferreira, Helder Silva (KEEP Solutions)

March 2011

This work was partially supported by the SCAPE Project. The SCAPE project is co-funded by the European Union under FP7 ICT-2009.4.1 (Grant Agreement number 270137).

Executive Summary

This report provides an overview and instructions for packaging software tools and environments in SCAPE. Software components like preservation tools will be packaged using the Debian/Redhat package management system contributing to a simplified and automatable installation procedure, and thus to the sustainability of the software product. Complex environments like the distributed execution platform and/or specific packaging-environments will be made available as EC2-compliant virtual machine images. This approach enables users to easily instantiate complex environments using a private and/or public cloud infrastructure from pre-configured VM images.

Table of Contents

1 Introduction

An important goal of the SCAPE project is the development of executable preservation workflows. Workflows are constructed based on the Taverna[1] workbench environment and typically rely on different preservation components. Components may be available as locally and/or remotely installed applications to the workflow designer. A major challenge is to make these workflows available on a scalable storage and execution environment. This in turn requires an automated strategy to resolve a workflow's dependencies on a distributed and dynamically scalable computing platform.

We believe that the employment of a packaging method will be paramount for the manageability and sustainability of the preservation components developed by the project. Consequently, the project has started to employ Linux (Debian, Redhat) software packaging and package management systems for SCAPE. Packaging provides a standardized and integrated way to manage and install software. A major benefit is the notion of *dependencies* which are specified by the software package and controlled by its package maintainer.

The major advantage of using packaged software is that the end-user can install a software bundle based on a single command (or a click using a package management GUI). The package manager will automatically download the requested package from a repository, resolve all dependencies, and install the software. This mechanism ensures a defined software installation process which can also be used to automatically update and/or remove packages.

Due to the diversity of SCAPE data sets, preservation workflows will depend on dozens of different preservation tools. Experimenters will rely on a variety of installations, both relying on Web services hosts as well as on a number of individually installed desktop computers, depending on the processed payload data. On the execution Platform, preservation tools must be provided dynamically on every node of the computational cluster, depending on a specific workflow. These requirements drive the need for a packaging model in SCPAPE in order to manage software and resolve dependencies between component developers, experimenters, and different execution systems.

In addition to packaging software components, SCAPE is making use of virtual machines in order to package complex environments. This allows us to provide SCAPE services as reusable images that can be launched on-demand using a private and/or public infrastructure like Amazon EC2. Providing a machine image has proven to be particularly useful for complex systems that require the installation and configuration of a number of software packages. For example, the SCAPE Platform is already a complex distributed system that requires significant effort and expertise in order to be properly installed and configured. However, provided as a pre-configured image, the Platform can be made available and easily deployed on an arbitrary number of nodes using a cloud system.

Packaging software properly is a non-trivial task as described in the Debian packaging guide below. However, once the environment required to create a particular software package has been set-up it can be used to generate a software package from its code repository. The packaging environment

[1] http://www.taverna.org.uk/

can be subsequently re-used in order to generate packages for future software releases with low configuration overhead. The OPF has made available a set of EC2 virtual machine images that are configured to automatically build software packages for different preservation tools. Their only function is to download the latest release (by tag number), build it and put it on the server's web page such that they can be downloaded.

2 Deploying Environments from Virtual Machines

2.1 Public/Private Cloud Infrastructures

SCAPE aims to employ virtual machine images and cloud environments in order to make complex systems that result from the project available. This allows users to easily carry out experiments and evaluate SCAPE environments (like the Platform). To this end, we have started to create images that are compliant with the Amazon EC2[2] and Open Eucalyptus[3] platforms.

Based on the Infrastructure-as-a-Services (IaaS) model, the cloud platforms allow users to lease configurable computer resources on demand based on a service API as well as a tools suite. AWS is a widely used cloud offering that provides access to computer resources distributed over different global regions. Eucalyptus provides a private cloud-computing platform that provides REST and SOAP interface compliance with EC2, S3, and EBS. Eucalyptus provides a highly scalable open-source cloud solution that can be set-up on a private infrastructure. Eucalyptus is the result of a research project at the University of California, Santa Barbara, and widely used in the academia.

2.2 Development Infrastructure at AIT

In the context of SCAPE, AIT has set up a Eucalyptus-based development cloud environment. The system presently provides 10 dual-core nodes (2 x 2.80 GHz) with 8GB of RAM and 2TB storage per node. The front-end (Debian Squeeze) hosts the Eucalyptus Cloud Controller, the Cluster Controller, and the Walrus storage service. It also provides a Fully Automated Installation (FAI) server. FAI is an automated installation framework that can be used to install Debian systems on a cluster, or number of different hosts. This allows us to easily add new nodes to the system, which can be booted via a network card using PXE, a pre-boot execution environment most modern network cards support. The cluster nodes run the XEN hypervisor and a Debian distribution that includes a Xen Dom0 kernel (*2.6.32*-5-*xen*-amd64). More information on installing the Xen virtual machine monitor on Debian Squeeze can be found on the Debian Wiki[4].

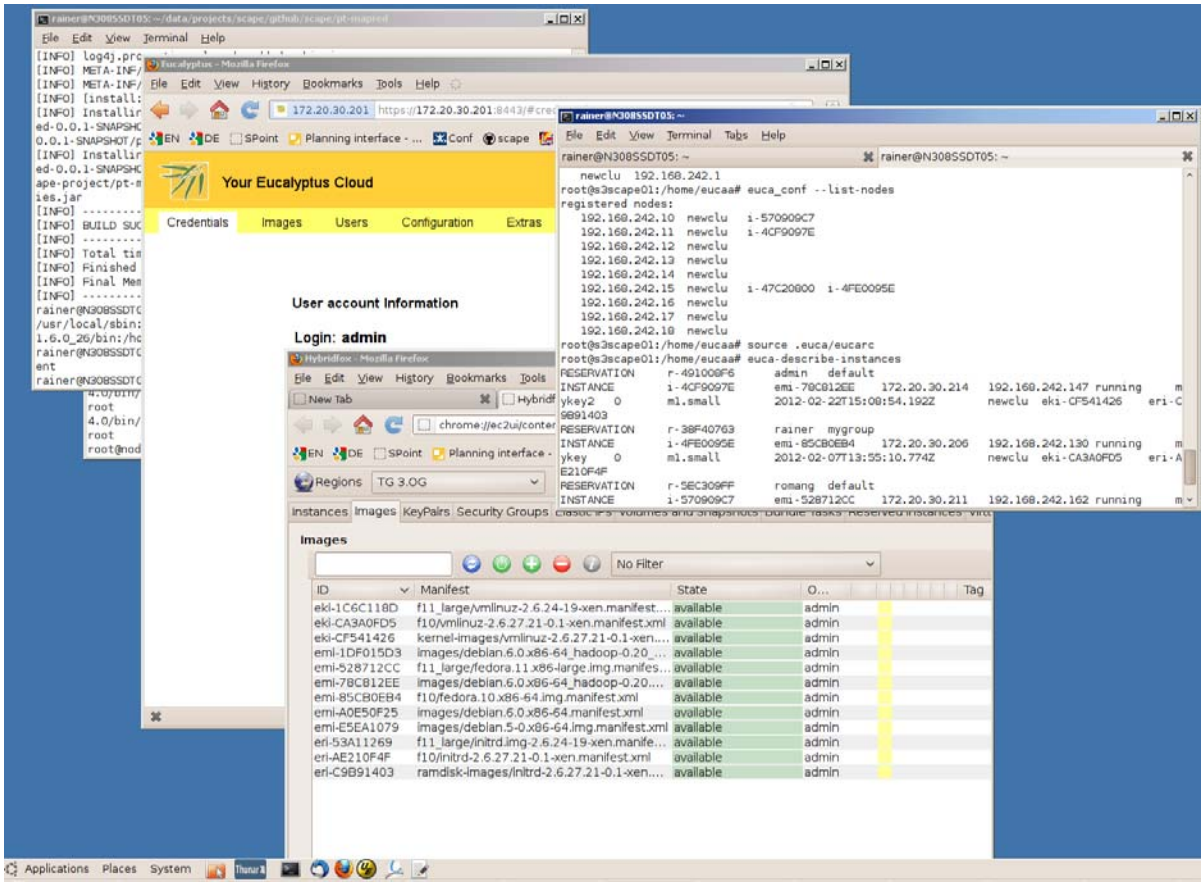
2.3 Deploying an Environment

Command-Line and Graphical Tools

The Amazon EC2 API provides a service interface to remotely control virtual machines instances allowing a user for example to discover, configure, and deploy them. Users can easily interact with

[2] http://aws.amazon.com/
[3] http://open.eucalyptus.com/
[4] http://wiki.debian.org/Xen

the SOAP/REST based services using the ec2-api-tools, a set of command-line tools. Amazon also provides a browser interface allowing one to control EC2 instances. Eucalyptus provides a similar command-line tool suite that is compatible with Amazon EC2. Tools are used for controlling and managing virtual machine instances, EBS volumes, elastic IPs, and security groups and can be used with both EC2 and Eucalyptus. Elasticfox[5]/Hybridfox[6] are browser add-ons for Mozilla Firefox that provide a very useful GUI abstraction for EC2/Euclyptus user account.



User-view on SCAPE development cloud at AIT: Eucalyptus web interface,
Hybridfox browser add-on, and terminal-based interaction.

User Identification and Authorization

Amazon's EC2 command-line tools demand a user to supply an X.509 certificate for querying and controlling Eucalyptus instances. The certificates can be downloaded from via the AWS/Eucalyptus web interfaces. Additionally an RSA key pair (public + private key) must be generated and registered with the cloud in order to identify a user against the virtual machine instances he/she launches. Setting-up the security credentials for an account is a prerequisite in order to interact with the cloud

[5] http://aws.amazon.com/developertools/609?_encoding=UTF8&jiveRedirect=1
[6] http://code.google.com/p/hybridfox/

environment, e.g. band on the command-line tool suite. Detailed instructions for setting up the EC2 API[7] and Euca2ools[8] command-line tool suite can be found online.

Using the Tool Suite to Deploy Virtual Machines

Virtual machines can be started, stopped, and access based a few commands described below. At the time writing these guidelines, the Amazon API/AMI tools (ec2-api-tools-1.3-30349, ec2-ami-tools-1.3-26357) are compatible with the Eucalyptus Euca2ools[9]. The following example shows the commands required to launch a virtual machine at AIT's Eucalyptus environment. The EC2 API tools can be used in a similar way to launch instances on EC2.

1) Show information about available machine images:

```
eucaa@s3scape01:~$ euca-describe-images

IMAGE   emi-EAD414E3    images/debian.6.0.x86-64_hadoop_demo.manifest.xml
admin   available       public           x86_64  machine
eri-C9B91403    eki-CF541426
IMAGE   emi-A0E50F25    images/fedora.10.x86-64.img.manifest.xml
admin   available       public          x86_64   machine
eri-C9B91403    eki-CF541426
```

2) Create new instance from image:

```
eucaa@s3scape01:~$ euca-run-instances -g demoGroup -k eucaaKey emi-EAD414E3

RESERVATION r-56560988      admin         demoGroup
INSTANCE    i-4E2A08E8      emi-EAD414E3  0.0.0.0     0.0.0.0
pending     eucaaKey        2012-03-15T10:02:42.233Z
eki-CF541426    eri-C9B91403
```

3) Show information about that instance:

```
eucaa@s3scape01:~$ euca-describe-instances | grep i-4E2A08E8

INSTANCE        i-4E2A08E8      emi-EAD414E3 172.20.30.206  192.168.242.130
running         eucaaKey 0      m1.small     2012-03-15T10:02:42.233Z
newclu  eki-CF541426    eri-C9B91403
```

4) SSH into the instance:

```
eucaa@s3scape01:~$ ssh -i eucaaKey.private root@192.168.242.130

root@debian:~#
root@debian:~# exit
```

5) Terminate the instance:

[7] http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/setting-up-your-tools.html
[8] http://open.eucalyptus.com/wiki/EucalyptusGettingStarted_v2.0
[9] http://open.eucalyptus.com/wiki/ec2-compatible-tools

```
eucaa@s3scape01:~$ euca-terminate-instances i-4E2A08E8
INSTANCE        i-4E2A08E8
eucaa@s3scape01:~$
```

Note: Virtual machine instances are associated with distinct security group (demoGroup in the above example). The Euca/EC2 tool suite provides a set of networking and security commands allowing a user to assign/unassign public IP address to instances, create security groups, and assign networking rules to them. By default, a VM instance denies incoming network traffic from all sources[10].

2.4 Pre-packaged Images and Compatibility

Eucalyptus provides pre-packaged images that are ready to use on a Eucalyptus cloud and that can be used as a basis to create customized virtual machine images. There is no fundamental difference between Amazon Machine Images (AMIs) and Eucalyptus Machine Images (EMIs). In order to deploy an AMI image on a Eucalyptus cloud, the image must be decrypted and bundled and encrypted with the Eucalyptus credentials. Additionally it is required to associate the disk image with the kernel and ramdisk modules used by the Eucalyptus cloud installation to make it an executable entity. More information on managing EMIs can be found on the Eucalyptus Community Wiki[11].

3 Software Packaging, Infrastructure and Projects

In the context of SCAPE, a number of projects and infrastructures exist to support developers and users of preservation tools in releasing software in the form of standardised packages. Enabling users to install, update and evaluate software through a simple and controlled process significantly contributes to the sustainability of a software product. SCAPE intends to primarily make use of the Debian package management system in order to distribute preservation tools and components.

3.1 Installing Packages

The Debian package management system consists of a stack of applications that handle downloading and installing Debian software packages. The Advanced Package Tool (APT) is capable of resolving dependencies between different packages. It retrieves the required packages from a Debian package repository and handles the installation and removal of the software. APT works together with *dpkg*, a lower-level tool which handles the actual installation and removal of packages. Aptitude is a text-based package manager for Debian systems that wraps the apt package management tool set[12]. Synaptic[13] provides a graphical package management program for apt. Apt-get (provided by APT) is a program to retrieve and install packages from multiple sources specified in */etc/apt/sources.list*. Apt-get (unlike *dpkg*) works based on package names and calls *dpkg* directly after downloading the archives file. A Debian repository provides an organized set of Debian packages. Once it has been added to the sources.list file, it allows a user to list and install all of the available packages, together

[10] http://open.eucalyptus.com/wiki/Euca2oolsNetworking
[11] http://open.eucalyptus.com/wiki/EucalyptusImageManagement_v1.5
[12] https://help.ubuntu.com/community/InstallingSoftware
[13] http://www.nongnu.org/synaptic/

with those provided by the Debian distribution. A reference on using the *apt-get*, *apt-cache*, and *aptitude* programs is included with the Debian documentation[14].

3.2 Building Packages based on Virtual Machines

Building software packages is a task which often requires a build environment that is heavily customized for the application being packaged. The resulting software package can then be installed on any machine using a package manager. OPF has started to making virtual machine images available on Amazon's utility cloud in order to provide reusable environments of this task. The images can be started up for very short amounts of time by in order to build and submit the latest version of their software to the central repository. These machines can also act as a good test environment and many machines can be started from the same image, reducing the IT overhead in maintaining physical machines.

These virtual machines are already set up to build a software package automatically. This is achieved by downloading the latest source code release from a repository (based on a tag number), building the software package, and putting it on the server's web page such that it can be downloaded. To carry out this process an Amazon AWS account is required to launch EC2 instances. A video showing the entire process as well as detailed instructions on building packages repeatedly using a virtual machine has been made available as part of the original blog post[15].

3.3 OPF Package Repositories

Debian/Ubuntu and RedHat/Fedora served as a starting point for several Linux distributions in using a packaging system. Software repositories provide users and distributors with control over the software that they are allowing to be installed on their system. OPF has set-up software package repositories for Debian (.deb)[16] and Fedora (.rpm)[17] packages to provide the package mirrors for each platform. This allows users to enable features like auto-upgrade that not only install each software package in one click, but also keep a system up-to-date with the latest releases.

3.4 Service for Change-Log Generation

In order to build a Debian package from source-code, it is required to provide a changelog file with the source code (as described in the packaging guidelines below). A changelog file record the changes made to a project and should be generated from a project's commit history. GitHub2ChangeLog (gh2ch) is a service that turns the GitHub commit history of a project into a fully structured changelog file, which is compatible with the stringent Debian packaging requirements. The service is currently available at http://p2-registry.ecs.soton.ac.uk/opf/gh2ch and as a Debian package on the OPF Debian package repository[18].

[14] http://www.debian.org/doc/manuals/debian-reference/ch02.en.html#_literal_apt_get_literal_literal_apt_cache_literal_vs_literal_aptitude_literal
[15] http://www.openplanetsfoundation.org/blogs/2012-03-08-turning-github-code-debian-packages-opf-way
[16] http://deb.openplanetsfoundation.org
[17] http://rpm.openplanetsfoundation.org
[18] http://deb.openplanetsfoundation.org/packages/gh2ch_1.0.0_all.deb

4 Debian Packaging Guidelines

4.1 Introduction

In order to increase the adoption and sustainability of the tools delivered, SCAPE is creating Debian packages that aid the end-user in installing and easily deploying these tools in literally hundreds of servers. In the following, guidelines for building SCAPE packages on a Debian system are provided. The guide is intended only as a reference for getting started. Once a user is comfortable with the tools and the challenges then the official maintainers guide[19] should be referred to! The latest version of these guidelines can be found online[20].

4.2 What One Must End Up With

A package is named similar to **package-name_1.3.2_amd64.deb.** In the following the individual parts that make up the file name are explained:

package-name: A short name which perfectly describes the package, e.g. droid, ffmpeg, vlc. The package name should be chosen so that it is searchable beyond the lifetime of the project and should not include any project names or names of technologies that are likely to disappear. The package name should be transferable to a future maintainer without needing to be changed.

1.3.2: The version number in X.Y.Z format conforms to the Debian and GNU version numbering standard of major.minor.revision. For clarity these are explained here:

X (major): Indicates a major release where the usage of the tool changes significantly, either via a user interface or its API. If the function of an existing API call has been change, then this version number MUST be incremented. It basically tells other systems that they may not work anymore.

Y (minor): Here, functionality and API calls can be added but existing ones MUST remain the same.

Z (revision): Used for security patches and bug fixes only! All Z revisions SHOULD relate to one or more tickets raised in an issue tracker.

_amd64: Indicates the supported architecture (other examples include i386 and others). Ideally tools which do not depend on an architecture should be distributed.

4.3 Pre-Requisites

As a pre-requisite, a Debian/Ubuntu machine which is capable of running the tool one wishes to package or compile is required. Ideally, this should be an Ubuntu LTS (10.04, 12.04 etc) machine which is most likely to be supported by the institutions who wish to run the tool.

[19] http://www.debian.org/doc/manuals/maint-guide/
[20] http://wiki.opf-labs.org/display/SP/Debian+Packaging

Note: If the tool is compiled with static libraries (e.g. from Python to C) then any Debian/Ubuntu distribution can be used to build the package, however testing the package to be distributed on a Ubuntu LTS version SHOULD be carried out.

On the build machine the following packages are required:

- build-essential
- dh-make
- devscripts
- debhelper
- lintian

These can be installed using the following command:

```
sudo apt-get install build-essential dh-make devscripts debhelper lintian
```

4.4 Getting Started

Before starting to build a package, one is typically confronted with either of 2 situations:

1. He/she has a piece of software that is already installable using ./configure, make and make install (i.e. it has a makefile). In this situation all that is needed is to carry out the so called "debianisation". In this situation the Debian maintainers guide[21] provides excellent guidelines which mainly describes this case.
2. The software does not have a makefile and has never been made installable. Here, it is important to clarify if it makes sense to have a makefile which can build and/or install the package. This way one can easily build and install it on most platforms that support *make*.

In either situation one will need to end up with a directory containing the application. This directory MUST be named conforming to convention for package names described above.

```
mkdir my-package_1.0.0
```

In order to start without having already packaged code (a tar ball with makefile etc) then it is required to build a **native** package. Otherwise the steps here stay the same:

```
cd my-package_1.0.0
dh_make --help
```

Follow the dh_make help text to fill in your name, email and any other details. Typically *a single package* (-s) will be built.

[21] http://www.debian.org/doc/manuals/maint-guide

Please note also that the package name should match the folder prefix (e.g. my-package). This should not be done later!

If this command executes correctly, it generates a *debian* folder containing a number of example files (explained in the maintainers guide). It is important to focus on the following four files:**Control**, **Rules**, **Copyright** and **Changelog**. The files are explained in more detail in the following section.

4.5 The Control File

This file provides information about the package to the packaging system. It contains the following information:

- Package Name
- Package Short Description
- Package Long Description

These fields are used to allow users to find the package among every other package. Therfore, it is important to choose the words carefully. It is recommended to ask how someone with no knowledge of the package might find it.

- Developer Name
- Homepage

The even more essential bits are:

- Compatible Architecture
    - Choose from ALL,ANY,i386,i686,amd64,ia64,sparc64....
- Build Dependencies (Build-Depends:)
- Run Dependencies (Depends:)

Note that the build dependencies could be completely different from the required run dependencies and thus are treated completely separate.

4.6 The Copyright File

The file provides an absolute requirement. It is important to make sure to separate the copyright of the upstream code (i.e. the program) from any copyright on the packaging files. If the upstream code is a combination of libraries then all copyright and licensing information for these must be detailed. It is recommended to use paths as shown in the example below:

```
library/droid
   Copyright: The National Archives (UK)
   License: .....
debian/*
   Copyright: ...
```

4.7 The Changelog File

If a revision control system is used properly (with branches and tags) then one should just be able to generate the changelog automatically from the commit history up until the tag that is currently packaged. The changelog should detail all changes to the upstream code, however it is often used to detail changes to the packaging files.

If the changelog is generated from the commit history, then the changelog should not be committed to the revision control system! As soon as it is committed, it is already out of date.

Note that the changelog is fussy in its format, but it is a format the *vim* editor recognises. In order to make the process of generating a changelog for a github hosted project easier, the OPF provides a GitHub 2 Changelog service (see section 3.4). Using this service one can simply add a call to this service to a rules file and download the changelog as part of the package building process.

4.8 The Rules File

This file is a makefile that should only contain Debian-specific make rules. Typically it is used to call *./configure*, *make*, and *make install* of the upstream code and provide pathing information in order to build a binary package ready for distribution.

Additionally however, the deb-helper scripts are also called from this file to automatically install items like documentation, manpages and set up any configuration interfaces for the package. These deb-helper scripts can (and should) also be used to verify that the system being used for both building and installing the package is in the correct state.

The following is an example install section:

```
binary-indep: install
    dh_testdir          <\- test required directories exist
    dh_testroot         <\- test the user has root privileges
    dh_installchangelogs <\- copy the changelogs to the correct location on whatever platform
    dh_installdocs      <\- copy the documentation to the correct location
    dh_installexamples  <\- copy the examples to the correct location
    dh_installman       <\- copy the man page to the correct location
    dh_installdebconf   <\- set up the configuration wizard for  the user
```

Note that these deb-helper scripts not only copy files to their correct location but also index them such that applications know the existence of the man page (for example) and can guide the user to this resource (e.g. through tab completion).

dh_installdebconf is perhaps the most obvious example of this. Deb Conf can be used to prompt the user for input whilst the package is being installed. Deb Conf automatically creates the interface in which this is done, e.g. via a terminal (ncurses) or via a full GUI.

4.9 Specifying a Manpage, Doc, Example, Etc

1. In the *debian* directory create a file called my-package [docs,manpages,examples].
2. List the paths of the files which are these in this file.

For example:

```
Getting_Started.pdf
debian/packaging_guide.pdf
```

These files are read by dh_install [docs,examples,man] and thus the files listed on each line are copied to the relevant place automatically.

4.10 Building the Package

Most commonly a package can be built with the following command (from the my-package_1.0.0 directory):

```
dpkg-buildpackage
```

Alternatively, to automatically build a version in git (from the directory managed by git) use the following command:

```
git-buildpackage
```

4.11 Building a Signed Debian Package

In order to build a fully signed Debian package it is required to first generate a key (if this has not been done previously). A good guide to generating a key can be found online[22]. Once a key is generated, it is important to **keep it save**.

To build a package with this key a command similar to the following is used:

```
dpkg-buildpackage –tc -k594F7FBA
```

Note that when building an *i386* and *amd64* version of a package, potentially on two different machines then it is required to check the hashes of the *.dsc* and *.tar.gz* files match in the two changes files! If they do not match the issue can be resolved by picking one of the *tar.gz*'s and *dsc*'s and matching the hashes to the values of one of the files.

Once this is done, remove the PGP sections at the top and bottom of the *.changes* file and then re-sign the changed file using the following command:

```
debsign -k594F7FBA my-package_1.0.0_i386.changes
```

[22] http://keyring.debian.org/creating-key.html

4.12 Verifying the Package

It is important to note **that no package will be accepted into any Debian Repository while it contains Lintian Errors**!

Lintian is the tool used to verify if a package is well formed (and not that it works). As well as looking at install paths, dependencies and well formed control files, lintian also checks spelling and other common errors that are made during the package building process.

Lintian only works on a built package and can be used by issuing the following command:

```
lintian my-package_1.0.0_all.deb
```

Output lines start with one of the following:

- I: Information Message
- E: Error (should be fixed)
- W: Warning (must be fixed)

More information on most of the errors and warnings can be found online[23].

5 The OPF Debian Repository

5.1 Adding the OPF Debian Repository

The following command adds the Open Planets Foundation repository to a Debian/Ubuntu machine. The command must be executed as root user.

```
add-apt-repository 'deb http://deb.openplanetsfoundation.org stable main'

apt-get update

apt-key adv --keyserver keys.gnupg.net --recv-keys A96D252D594F7FBA

apt-get update
```

5.2 Submitting a Package to the OPF repository

In order to upload software to the OPF repository the *dupload* package is required:

```
sudo apt-get install dupload
```

[23] http://lintian.debian.org/tags-severity.html

First it is required to edit the dupload.conf file:

```
sudo vim /etc/dupload.conf
```

Find the *$default_host* line and change it to "opf" as follows:

```
$default_host = "opf";
```

Ensure that the following lines are un-commented:

```
$preupload{'changes'} = '/usr/share/dupload/gpg-check %1';
$preupload{'deb'} = 'lintian -v -i %1';
```

Then add the OPF endpoint to the configuration by adding the following:

```
$cfg{'opf'} = {
        fqdn => "deb.openplanetsfoundation.org",
        incoming => "incoming/",
        dinstall_runs => 1,
};
```

Save this file and return the command line, from here make sure to enable FTP_PASSIVE mode: By default reprepro will not ask for a passphrase. With all this done the file can be uploaded:

```
dupload package-name_1.0.0_all.deb
```

In case of a port error, ensure that passive mode is enabled.

Finally, you are responsible for ensuring all files are uploaded to the server and your package will appear once it has been checked, if clean it should appear within an hour. The installation packages will be available at the OPF package repository at http://deb.openplanetsfoundation.org/.

5.3 The OPF Debian Repository (Server Setup)

The following provides a brief overview of the configuration of the OPF Debian repository server. The Debian package repository has been built based on guideline available online24.

The distributions file of the repository:

```
Origin: Open Planets Foundation
Label: Debian Repository
Suite: stable
Codename: stable
Version: 3.1
Architectures: i386 amd64 all source
```

[24] http://www.debian-administration.org/articles/286

```
Components: main non-free contrib
Description: Debian Repository for the Open Planets Foundation
SignWith: 594F7FBA

Origin: Open Planets Foundation
Label: Debian Repository
Suite: unstable
Codename: unstable
Architectures: i386 amd64 all source
Components: main non-free contrib
Description: Debian Repository for the Open Planets Foundation
SignWith: 594F7FBA
```

The ask-passphrase option should be set in the conf/options file to use a GPG key that requires a password. By default reprepro will not ask you for your passphrase.

The options file:

```
ask-passphrase
```

 In order to make the reprepro work the OPF Repository Private GPG Key (that has very restricted access) is needed. One can create his/her own key and change the SignWith line above.

The FTP server is vsftpd with the following config file in /etc/vsftpd.conf:

```
listen=YES
anonymous_enable=YES
local_enable=NO
write_enable=YES
anon_upload_enable=YES
dirmessage_enable=YES
xferlog_enable=YES
connect_from_port_20=YES
chown_uploads=YES
chown_username=davetaz
anon_root=/home/deb.openplanetsfoundation.org
chroot_local_user_YES


pasv_min_port=10000
pasv_max_port=10021


secure_chroot_dir=/var/run/vsftpd
pam_service_name=vsftpd
rsa_cert_file=/etc/ssl/certs/ssl-cert-snakeoil.pem
rsa_private_key_file=/etc/ssl/private/ssl-cert-snakeoil.key
```