Innopolis University

SYSTEM AND NETWORKING ENGINEERING

Essential Skills

LABORATORY REPORT 3

Build systems (Autotools)

Student Name	**Student ID**
Sergey Grebennikov	47611

Lecturer:

Stanislav Litvinov

Submission Date : January 22, 2019

Contents

1 Introduction

The core parts of a Linux distribution and most of its add-on software are installed via the Packaging System. The packaging system allows to install, remove, and build packages. A package refers to a compressed file archive containing all of the files that come with a particular application. The aim of packaging is to allow the automation of installing, upgrading, configuring, and removing computer programs for system.

Each package contains the files and other instructions needed to make one software component work on the system. Packages can depend on each other and will not install unless that package is also installed at the same time.

There are two broad families of package managers: those based on Debian and those which use RPM as their low-level package manager. The two systems are incompatible, but provide the same features at a broad level. [1] [2]

Lab work consists of two assignments:

1. Packaging systems
2. Installing a game

2 Main Part

2.1 Packaging systems

1. Study the Debian packaging system and answer the following questions:

   - How does it work?

     Answer:

     The Debian packaging system provide two packaging tool levels: a low-level tool **dpkg**, takes care of the details of unpacking individual packages, running scripts, getting the software installed correctly, while a high-level tool **apt-get** (**A**dvanced **P**ackage **T**ool) works with groups of packages, downloads packages from the vendor, and figures out dependencies. (Figure 1)

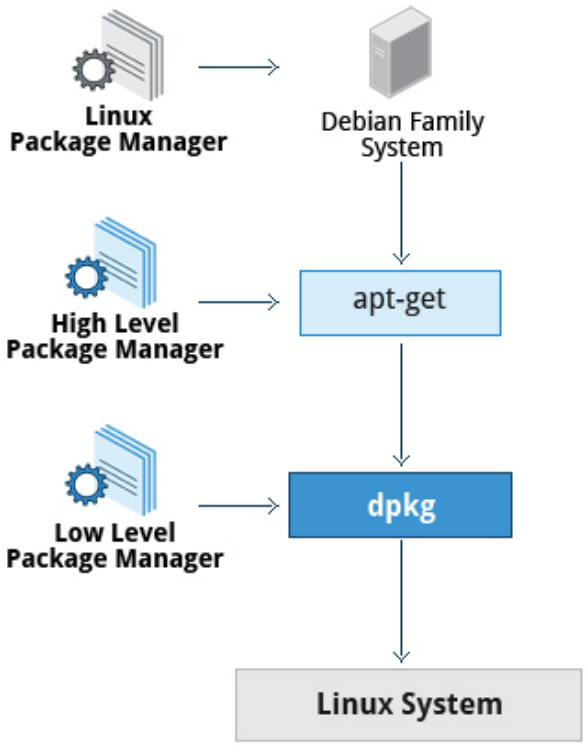

Figure 1: Debian package managers

A Debian Package is a file that ends in *.deb* and contains software for Debian system. A *.deb* is also known as a *binary package* and is an **ar** archive. This means that the program inside the package is ready to run on the system. There are also *source packages*. Source packages provide all of the necessary files to compile or otherwise, build the desired piece of software.

A Debian Package is comprised of three files (Figure 2):

- **debian-binary**. It indicates the version of the .deb file used.
- **control.tar.**gz. It contains the meta-information.
- **data.tar.**xz. It contains the files to be extracted from the package.

A source package is usually comprised of three files, a **.dsc**, a **.orig.tar.gz**, and a **.debian.tar.gz** (or **.diff.gz**). They allow creation of binary packages (**.deb** files described above) from the source code files of the program, which are written in a programming language. (Figure 3)

```
sergey@gsa-sne:~/Downloads$ ar t google-chrome-stable_current_amd64.deb
debian-binary
control.tar.gz
data.tar.xz
```

Figure 2: The Debian Package

```
sergey@gsa-sne:~$ ls
anaconda3  gnujump-1.0.8                  gnujump_1.0.8.orig.tar.gz  Public
Desktop    gnujump_1.0.8-1_amd64.build    gnujump-1.0.8.tar.gz       Templates
Documents  gnujump_1.0.8-1.debian.tar.xz  Music                      Videos
Downloads  gnujump_1.0.8-1.dsc            Pictures
```

Figure 3: The Source Package

- How does it deal with dependencies?

  Answer:

  Installation of software by the package system uses "dependencies" which are designed by the package maintainer. The **dpkg** does not automatically download and install packages nor does it satisfy their dependencies, in contrast to **apt**.

  The information about dependencies is stored in the file which contains "meta-information" (**control.tar.gz**). The dependencies are defined in the Depends field in the package header (Figure 4).

```
sergey@gsa-sne:~/Downloads$ cat `tar tzf control.tar.gz | grep "control"`
Package: google-chrome-stable
Version: 60.0.3112.101-1
Architecture: amd64
Maintainer: Chrome Linux Team <chromium-dev@chromium.org>
Installed-Size: 237182
Pre-Depends: dpkg (>= 1.14.0)
Depends: gconf-service, libasound2 (>= 1.0.16), libatk1.0-0 (>= 1.12.4), libc6 (
>= 2.15), libcairo2 (>= 1.6.0), libcups2 (>= 1.4.0), libdbus-1-3 (>= 1.1.4), lib
expat1 (>= 2.0.1), libfontconfig1 (>= 2.11), libgcc1 (>= 1:4.1.1), libgconf-2-4
(>= 3.2.5), libgdk-pixbuf2.0-0 (>= 2.22.0), libglib2.0-0 (>= 2.28.0), libgtk-3-0
 (>= 3.9.10), libnspr4 (>= 2:4.9-2~), libpango-1.0-0 (>= 1.14.0), libpangocairo-
1.0-0 (>= 1.14.0), libstdc++6 (>= 4.8.1), libx11-6 (>= 2:1.4.99.1), libx11-xcb1,
 libxcb1 (>= 1.6), libxcomposite1 (>= 1:0.3-1), libxcursor1 (>> 1.1.2), libxdama
ge1 (>= 1:1.1), libxext6, libxfixes3, libxi6 (>= 2:1.2.99.4), libxrandr2 (>= 2:1
.2.99.3), libxrender1, libxss1, libxtst6, ca-certificates, fonts-liberation, lib
appindicator1, libnss3 (>= 3.17.2), lsb-release, xdg-utils (>= 1.0.2), wget
Provides: www-browser
Section: web
Priority: optional
Description: The web browser from Google
 Google Chrome is a browser that combines a minimal design with sophisticated te
chnology to make the web faster, safer, and easier.
```

Figure 4: Dependencies

- Does it use the GNU build tools? How?

  Answer:

  Yes. The GNU build system is a free software and open source package that can be used in the Debian family system. The GNU build system consists of the GNU utility programs *Autoconf*, *Automake*, and *Libtool*. *Autoconf* generates a configure script. *Automake* helps to create portable Makefiles. *Libtool* helps manage the creation of static and dynamic libraries on various Unix-like operating systems.

  But in addition to the GNU build system, the Debian family has built-in tools for

building packages. Firstly it is necessary to get a copy of the upstream[1] software, generally in a compressed tar format. Then add modification to the upstream software under the debian directory, and create non-native source package in 3.0 (quilt) format. And finally build Debian binary packages. (Figure 5)

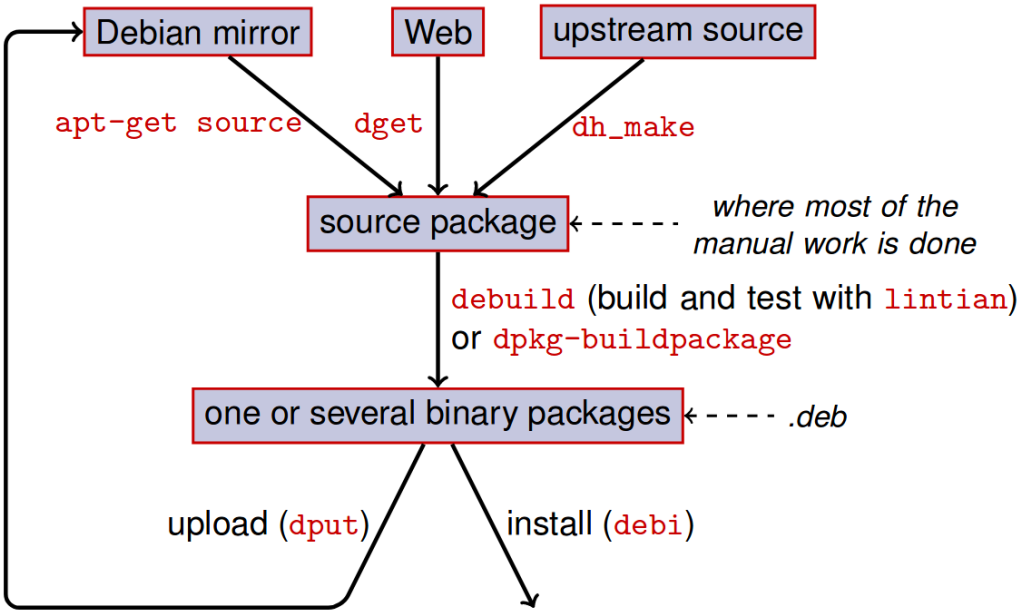

Figure 5: General packaging workflow

2. Study the Open SUSE packaging system and answer the following questions:

   - How does it work?

     Answer:

     The Open SUSE packaging system uses package management system to install, upgrade and remove software on the system. Software are distributed through Packages that contains a metadata and a list of dependencies. Packages are provided by repositories. (Figure 6)

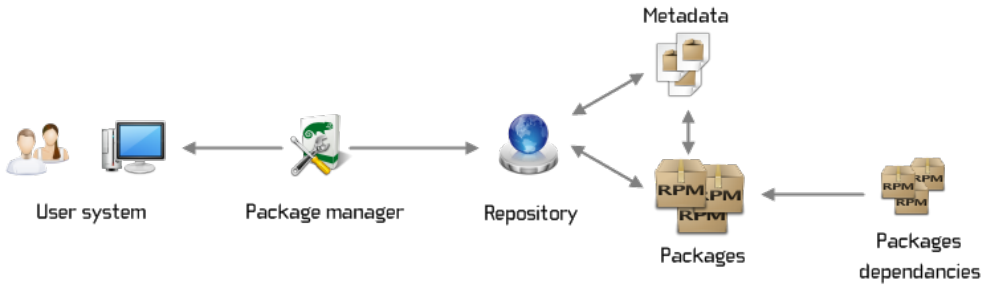

Figure 6: Open SUSE packaging system

The Open SUSE package management system provide two packaging tool levels: a low-level tool **rpm** is pre-compiled archives, and a high-level tool **zypper** (or

[1]In software development, upstream refers to a direction toward the original authors or maintainers of software that is distributed as source code, and is a qualification of either a bug or a patch.

YaST Software Management in GUI) which works on top of RPM, gets software packages from repositories, solves the dependencies and installs them on the system.(Figure 7)

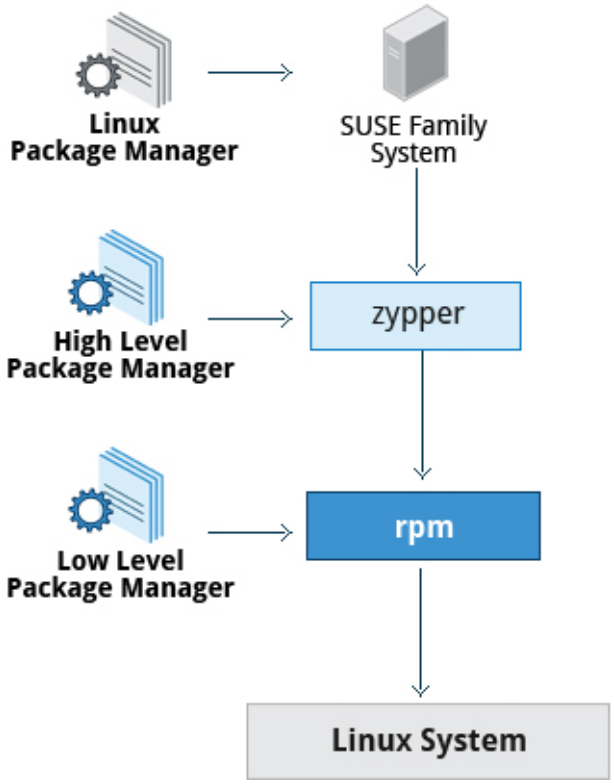

Figure 7: Open SUSE package managers

- How does it deal with dependencies?
  Answer:
  When packages are created they provide a so called **.spec** file. A spec file defines the attributes of the package, explicit dependencies, and how the content of the package is created.
- Does it use the GNU build tools? How?
  Answer:
  Yes. The GNU build system is a free software and open source package that can be used in the openSUSE family system. The GNU build system consists of the GNU utility programs *Autoconf*, *Automake*, and *Libtool*. *Autoconf* generates a configure script. *Automake* helps to create portable Makefiles. *Libtool* helps manage the creation of static and dynamic libraries on various Unix-like operating systems.
  But in addition to the GNU build system, the SUSE family has built-in tools for building packages from source packages. A source package can build binary with the **rpmbuild** tool. It requires the spec file to be in a specific location.
- If not what does it use?
  Answer:

2.2 Installing a game

1. Look for a small and simple game on the web

- Download its source package for Ubuntu (debian package? **Debian Source package**)

```
$ apt-get source bastet
```

- Create a binary from the source for the desktop (Figure 8)

```
$ cd bastet-0.43/
$ sudo apt-get build-dep bastet
$ debuild -b -uc -us
$ cd ..
$ sudo dpkg -i bastet_0.43-4build1_amd64.deb
$ bastet
```

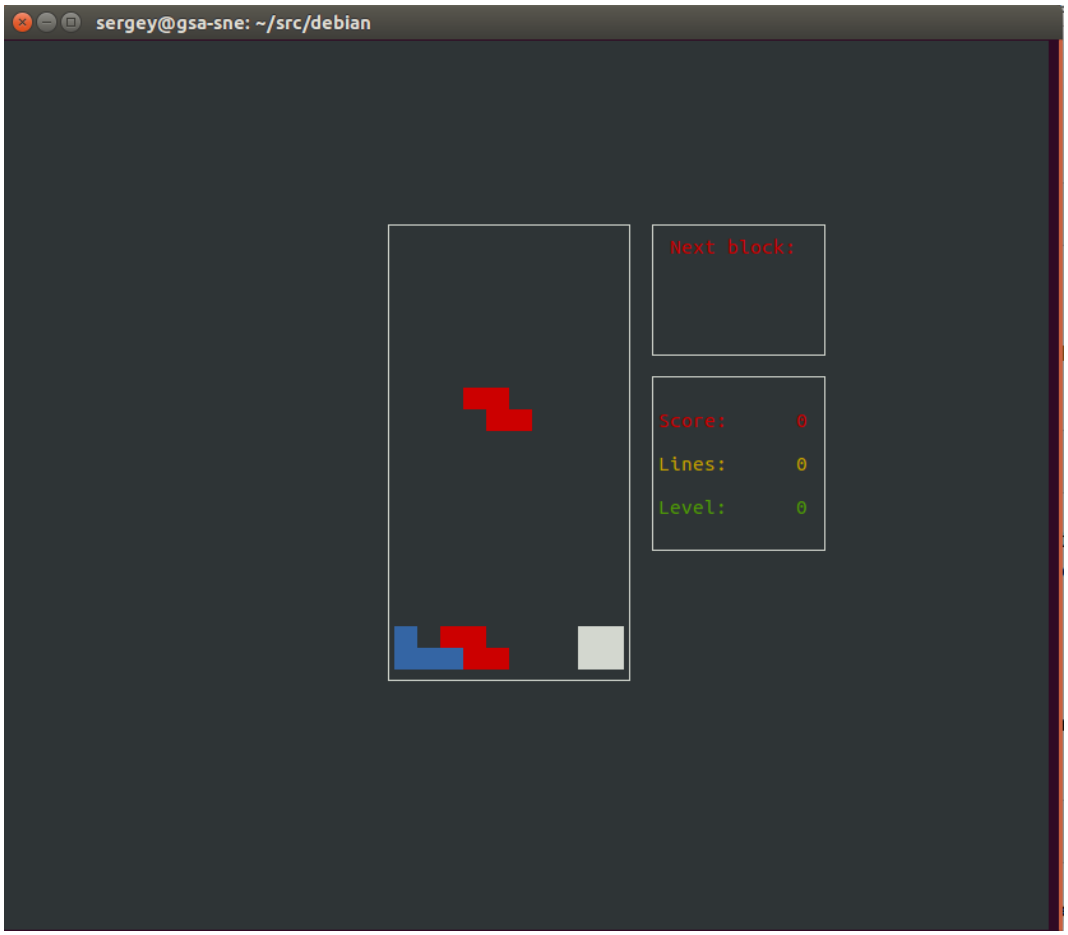

Figure 8: Bastet ("bastard Tetris")

- Create a binary from the source for the i386 architecture

```
$ sudo apt source bastet
$ cd bastet-0.43/
$ sudo pbuilder --create --architecture i386
$ sudo pbuilder --update
$ sudo pbuilder --build ../bastet_0.43-4build1.dsc
$ sudo dpkg -i /var/cache/pbuilder/result/bastet_0.43-4build1_i386.deb
$ file /usr/games/bastet
/usr/games/bastet: setgid ELF 32-bit LSB executable, Intel 80386, version 1 (GNU/Linux),
dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.32,
BuildID[sha1]=ebfa80b80166c0fa3bfbf43df56b6a3139d32a6d, stripped
```

3 Conclusion

The core parts of a Linux distribution and most of its add-on software are installed via the Packaging System. There are two broad families of package managers: those based on Debian and those which use RPM as their low-level package manager. The two systems are incompatible, but provide the same features at a broad level.

There are many *nix systems and they all differ slightly. The GNU build system (or Autotools) is needed so that the same source packages can be compiled on any *nix system.

4 References

[1] Package Management Systems on Linux https://courses.edx.org/courses/course-v1:LinuxFoundationX+LFS101x.2+1T2015/courseware/.

[2] Debian. Packaging https://wiki.debian.org/Packaging#Packaging_Procedures.

[3] The Debian Administrator's Handbook https://debian-handbook.info/browse/stable/sect.source-package-structure.html

[4] Package: dpkg-dev (1.18.24) https://packages.debian.org/stretch/dpkg-dev

[5] Wikipedia: GNU Build System https://en.wikipedia.org/wiki/GNU_Build_System

[6] Wikipedia: dpkg https://en.wikipedia.org/wiki/Dpkg

[7] Basics of the Debian package management system https://www.debian.org/doc/manuals/debian-faq/ch-pkg_basics.en.html

[8] Debian New Maintainers' Guide https://www.debian.org/doc/manuals/maint-guide/index.en.html

[9] Debian Packaging Tutorial http://www.debian.org/doc/manuals/packaging-tutorial/packaging-tutorial.en.pdf

[10] openSUSE:Packaging guidelines https://en.opensuse.org/openSUSE:Packaging_guidelines

[11] Debian Games https://blends.debian.org/games/tasks/