BEGINNER'S TO GUIDE
UBUNTU PACKAGE MANAGEMENT

BASIC / DPKG / APT-GET / SOURCES.LIST / REPOSITORY / PPA

an ebook from ubuntubuzz!

ADE MALSASA AKBAR
JANUARY 2017

Ebook Details

Title: Beginner’s Guide To Ubuntu Package Management
Author: Ade Malsasa Akbar
Publish date: January 25th 2017

Difficulty: beginner
Pages: 38
Fonts: Liberation Serif, Liberation Sans, Liberation Mono
Production: this ebook produced with LibreOffice on Ubuntu
Original website: http://ubuntubuzz.com

Author’s Preface

Dear all UbuntuBuzz readers,

Thank you for reading this free ebook! This ebook is a collection of 4 series of articles in UbuntuBuzz website about Ubuntu Package Management. This ebook is written for beginners so that they can do daily tasks by just reading a short ebook. I hope this ebook will be very easy to understand for anyone. However, the original sources of 4 series are here:

1. http://www.ubuntubuzz.com/2016/05/ubuntu-package-management-part-1-dpkg.html
2. http://www.ubuntubuzz.com/2017/01/ubuntu-package-management-part-2-basic-apt-get-commands.html
3. http://www.ubuntubuzz.com/2017/01/ubuntu-package-management-part-3-basic-sourceslist-settings.html
4. http://www.ubuntubuzz.com/2017/01/ubuntu-package-management-part-4-ppa-and-third-party-repository.html

Hereby, I express a big gratitude to the UbuntuBuzz owner, Mr. Mahmudin Ashar. Without Mr. Mahmudin Ashar’s guidance, I can not push further this beginner’s guide ebook about Ubuntu.

Finally, I apologize for any mistake in this ebook, I hope anyone can come to give feedback and correction. I am glad if anyone can contact me by email to teknoloid@gmail.com. Above it all, I wish this book will be useful for anyone of you.

Best regards,

Ade Malsasa Akbar

Table of Contents

Part 1: Basic Dpkg Commands

This tutorial introduces how to use **dpkg** package manager by command lines. This article is intended for beginners who just started Ubuntu. It consists of listing, installing, removing, and purging. It including some examples too. And this article is compatible with any Debian family operating system (such as Linux Mint) too. We hope this helps.

Reading Conventions

- Any command preceded with a '**#**' is should be run as root (or using sudo), and with a '**$**' is should be run as normal user.
- You may change **<package_file_name>** variable into one package file name, or more than one package file names separated by spaces, or type one `*` (asterisk) character to mean all packages inside current directory.
- This article assumes dpkg runs in the same directory with the package/packages.
- Don't install package blindly e.g. don't install vivid package in trusty, only install trusty package for trusty, and don't install package from any untrusted source. This may compromise your system stability and security.

1. List Installed Packages

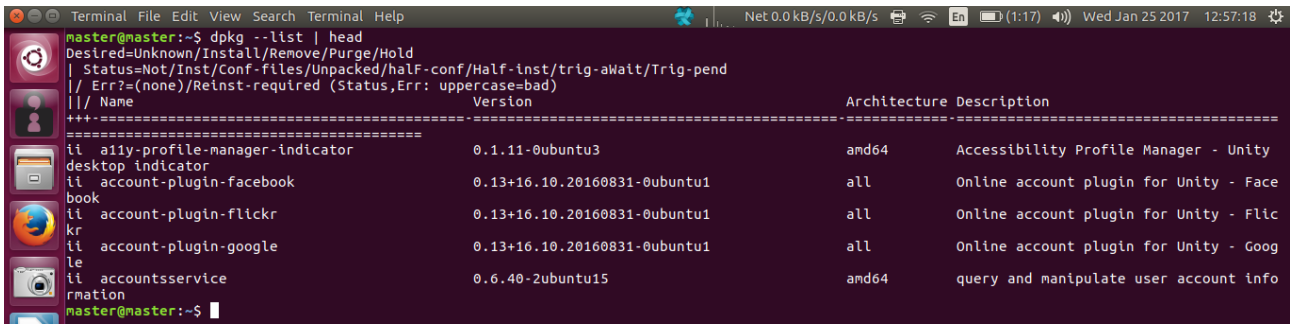
Command synopsis:

```
$ dpkg --list
$ dpkg --list <package_name>
$ dpkg -l <package_name>
$ dpkg --list | grep -i firefox
```

Example 1:

```
$ dpkg --list
```

Output 1:



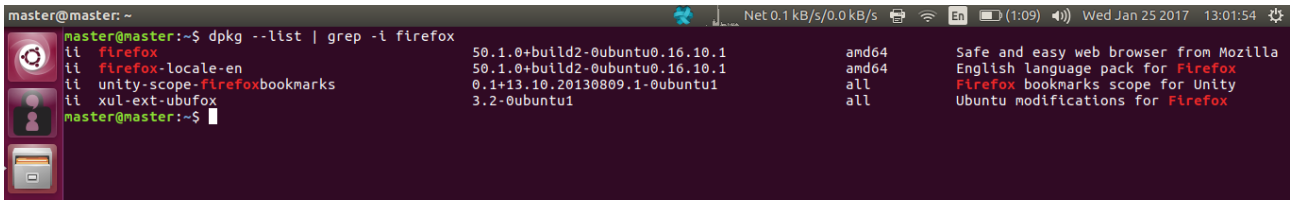
Explanation 1:

The **--list** option is used to show all of dpkg database contents sorted by package names. It shows all installed packages list, along with their statuses, versions, and descriptions information. This command can be specified for particular package by typing its name as argument like the third command. The output example shows how normal output looks like. Because an Ubuntu system usually consists of thousands of installed package, the output will be very long.

Example 2:

```
$ dpkg --list | grep -i firefox
```

Output 2:



Explanation 2:

This command line composed from 2 commands, *dpkg* and *grep*. The *dpkg --list* shows all packages installed and the *grep* filters the result according to a “search keyword”. So in this example the long result is cut into only packages that have “firefox” keywords in their names or their description. Imagine “googling” for your own installed packages in your own computer. If you want to search for another package, just change the keyword to anything else. This command line is very useful when you need to search installed packages one by one.

2. Install Package

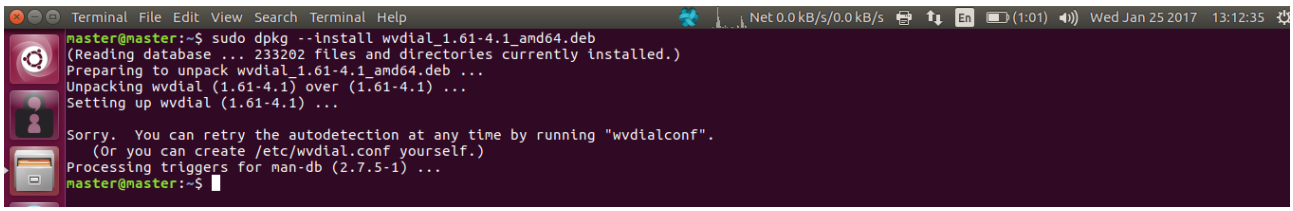
Command synopsis:

```
# dpkg --install <package_file_name>
# dpkg -i <package_file_name>
# dpkg --install wvdial_1.61-4.1_i386.deb
# dpkg --install *
```

Example:

```
# dpkg --install wvdial_1.61-4.1_i386.deb
```

Output:



Explanation:

The **--install** option tells dpkg to install a package or some packages. The most convenient way to install package is by installing package belongs to the same directory with your console. First you should make sure your console belongs to the same directory with the package (use **pwd** and **ls** command to know), then you should type the correct name of package you want to install. Third command example and the output above show you how to do it.

3. Remove Package

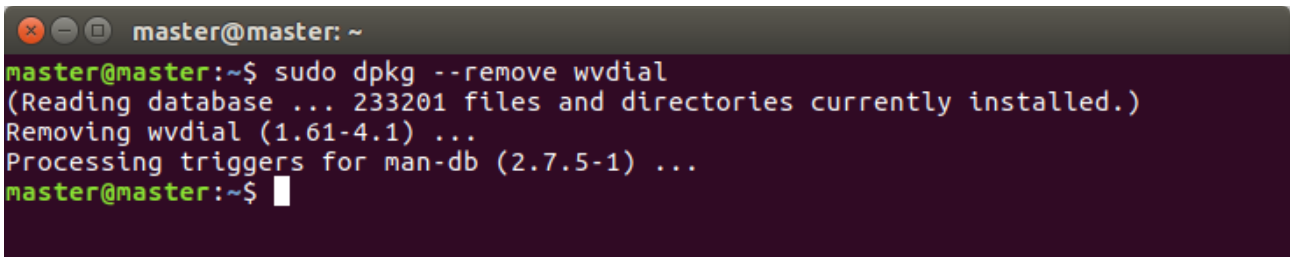
Command synopsis:

```
# dpkg --remove <package_name>
# dpkg -r <package_name>
# dpkg --remove wvdial
```

Example:

```
# dpkg --remove wvdial
```

Output example:



Explanation:

The --**remove** option tells dpkg to remove one or some installed packages. To remove you can do it in any directory, but you should type the **package_name** instead of **package_file_name**. That's why third command example shows **wvdial** name instead of **wvdial_1.61-4.1_i386.deb** file name.

4. Purge Package

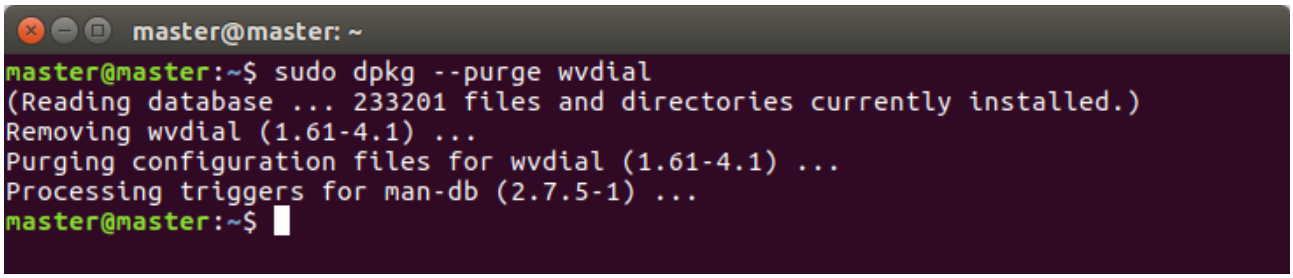
Command synopsis:

```
# dpkg --purge <package_name>
```

Example:

```
# dpkg --purge wvdial
```

Output example:



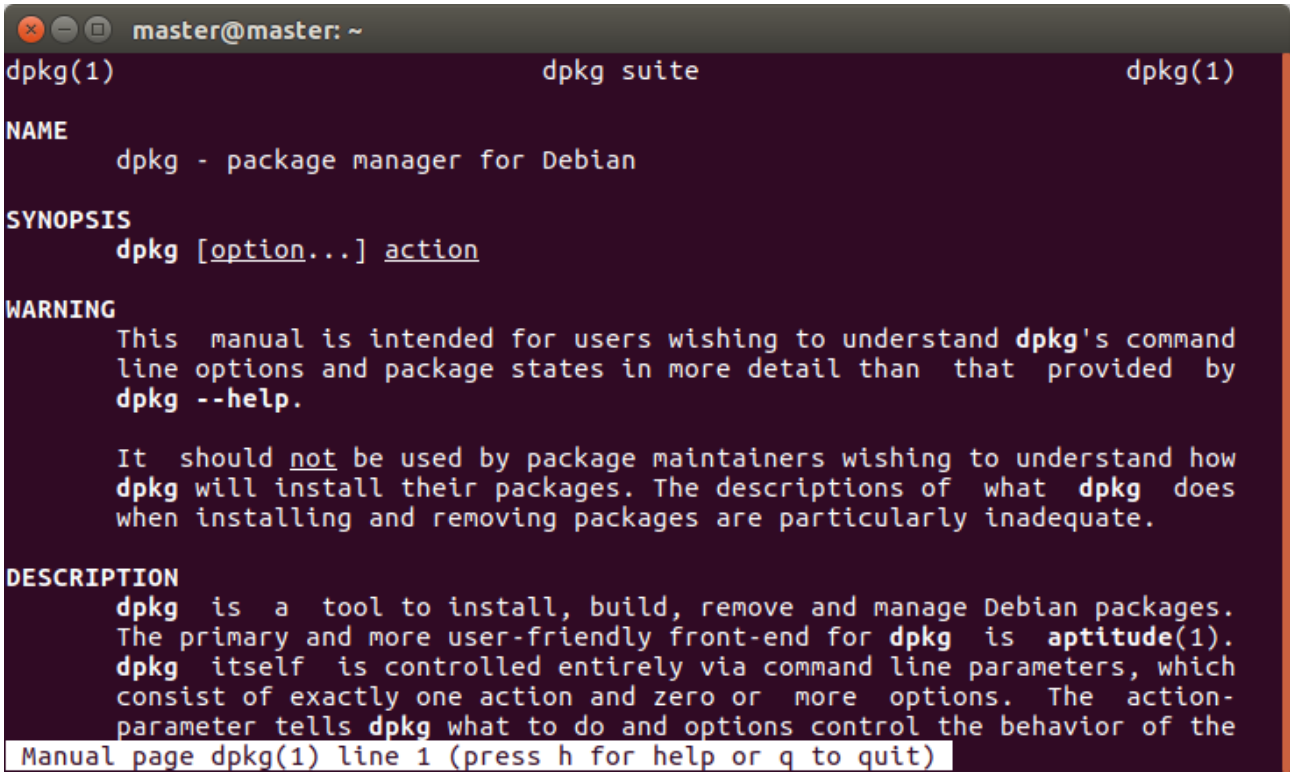
Explanation:

The --**purge** option tells dpkg to remove one or some packages. Purge is same as remove, but purge deletes also installed configuration files of the package. Notice the difference from output example while --purge shows "*Purging configuration file for wvdial*" message.

Getting Help

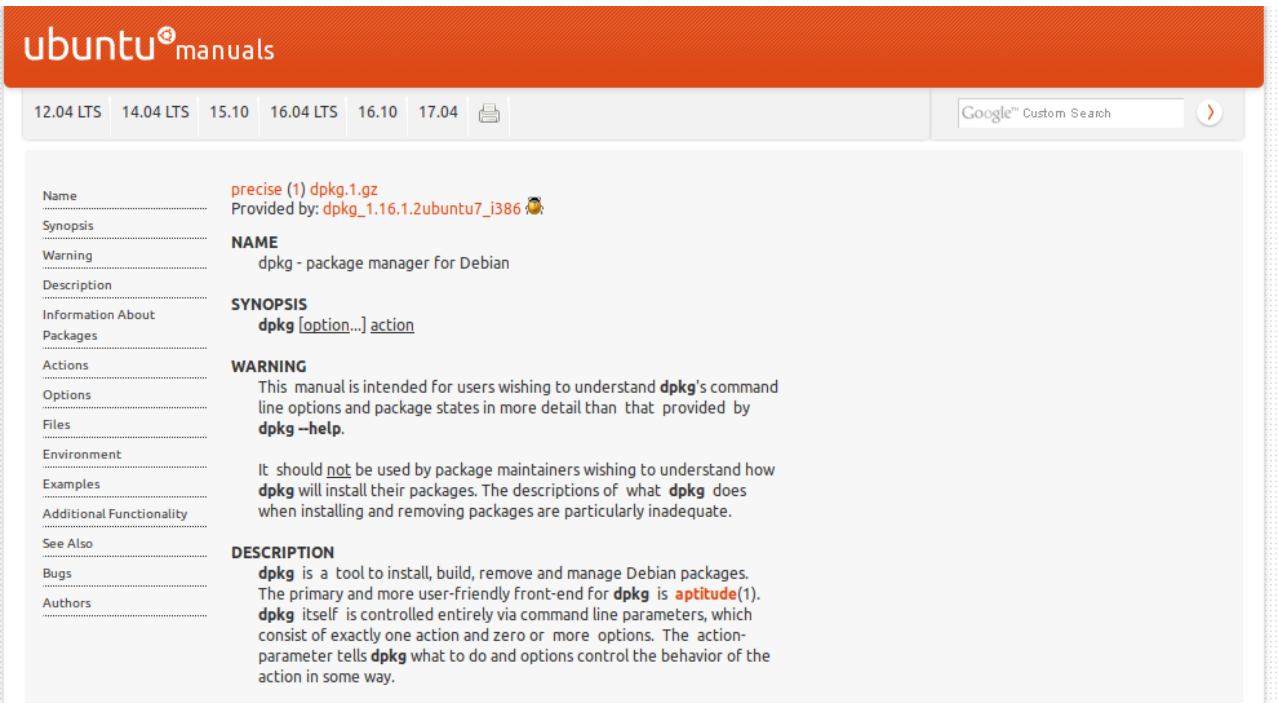
Internal:

Type command `$ man dpkg` to show dpkg manual documentation via Terminal.



External:

Read dpkg manual documentation online on the web such as http://manpages.ubuntu.com/manpages/precise/en/man1/dpkg.1.html.


ubuntu manuals
12.04 LTS
14.04 LTS
15.10
16.04 LTS
16.10
17.04
Google™ Custom Search
Name
Synopsis
Warning
Description
Information About Packages
Actions
Options
Files
Environment
Examples
Additional Functionality
See Also
Bugs
Authors
precise (1) dpkg.1.gz
Provided by: dpkg_1.16.1.2ubuntu7_i386
NAME
dpkg - package manager for Debian
SYNOPSIS
dpkg [option...] action
WARNING
This manual is intended for users wishing to understand dpkg's command line options and package states in more detail than that provided by dpkg --help.
It should not be used by package maintainers wishing to understand how dpkg will install their packages. The descriptions of what dpkg does when installing and removing packages are particularly inadequate.
DESCRIPTION
dpkg is a tool to install, build, remove and manage Debian packages. The primary and more user-friendly front-end for dpkg is aptitude(1). dpkg itself is controlled entirely via command line parameters, which consist of exactly one action and zero or more options. The action-parameter tells dpkg what to do and options control the behavior of the action in some way.

Part 2: Basic Apt-Get Commands

This beginner's guide for Part 2 explains **Apt-get** commands with examples for Ubuntu package management. This including some required explanations (about package, how apt works, and apt "database") and screenshots for 10 examples. This guide also includes *add-apt-repository* command, despite it's not part of APT, it's mentioned here to ensure beginner users happy with PPA and third-party repositories. This guide is a continuation of the Part 1 Dpkg Commands. I hope everyone can take advantage from this article. Enjoy.

Package

In Ubuntu operating system platform, software is “distributed” by Ubuntu developer to all the users. Ubuntu already distributes more than 80.000 software at the 16.10 version at 2016. Software is distributed in certain file called “package”. The user install software in their Ubuntu system also in form of “package”. So in other words working with Ubuntu is working with software packages.

Ubuntu distribution gives two types of package to the users, **source code** package and **binary** package. Source package is a file with **.tar.gz** format while binary package is a file with **.deb** format. Ubuntu developer puts large efforts to transform all source packages into binary packages so every users can just install the .deb instantly and run the software in their computer. All software packages of Ubuntu distribution stored at public server named **repository**. Every Ubuntu user installs software from repository.

How APT Works

APT is a complex system composed of apt-get many other apt programs. But the main concept is simple, this APT system is basically a client of Ubuntu repository server. In other words, APT is a local program in your Ubuntu system to use a repository and then to install, remove, or upgrade packages from that repository into your system. In non-technical sense, APT is a system to manage packages.

In technical sense, APT works for repository and anything related to repository. To fulfill this job, APT works cooperatively with Dpkg, making use of both databases to know internal and external packages, thus doing package management in your local Ubuntu system. APT’s main purpose is *to resolve dependency*, and that’s why people calls APT *dependency resolver*. APT works by reading Ubuntu internal databases to determine what packages installed and what packages not installed, then doing a calculation resulting complete list of packages need to be installed, and then download those packages from repository through the network. The final job is actually done by Dpkg to

install packages one by one. APT does not install package, APT is just *dependency resolver.* By downloading, APT stores all packages downloaded at **/var/cache/apt/archives/** directory.

In order to work with repository, of course, APT needs the address of the repository itself. This requirement is fulfilled by **sources.list** settings. The sources.list is actually a text file containing URL addresses of repository and some "codes" to determine what "room" of repository enabled/disabled. The settings is located in **/etc/apt/sources.list** file. APT system gives more options by providing /etc/apt/sources.list.d/ directory if the user wants to use third-party repository.

Important Database

APT works with its own "database" in your Ubuntu system located at **/var/lib/apt/lists/**. APT database is basically a bunch of text files storing complete information information such as name, URL, size, dependencies, description, etc. of every single of thousand packages from the repository. To make APT works, first the user must invokes APT to download the "database" from repository (I call it here: *reload*). Once this database obtained, then the user can perform search for any package they want and ask APT to install any package. Without this database APT will not work.

1. Reload

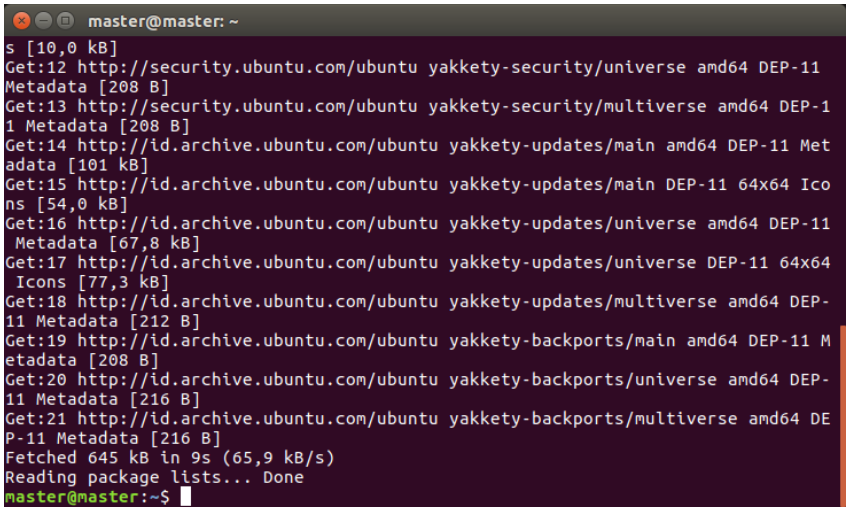
Command synopsis:

```
$ sudo apt-get update
```

Example:

```
$ sudo apt-get update
```

Output:



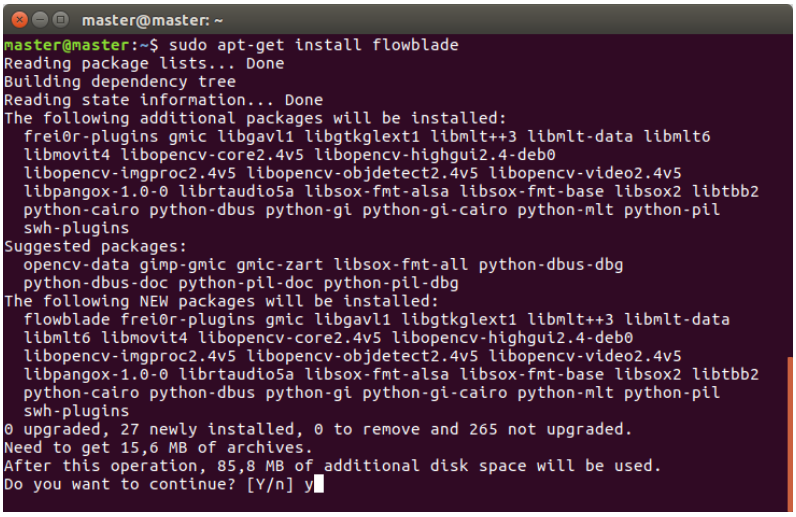
Explanation:
The *update* command downloads repository index. This command connects directly into sources.list settings in your system. Repository index is database files containing complete information about the repository and all packages inside it. You may consider it as the “map of repository”. Whenever you change your sources.list settings, you should run this update command once. Without this “map”, APT can not download any package.

2. Install

Command synopsis:
```
$ sudo apt-get install <package_name>
```

Example:
```
$ sudo apt-get install flowblade
```

Output:



Explanation:
By using this command line, you install a software from repository. You will be asked firstly for password, type it anyway and press Enter; then you will be asked secondly for permission, type “y” letter anyway and press Enter. The APT will download all packages needed and the software will be installed in your system. Do not close the terminal while the installation progress is still going.

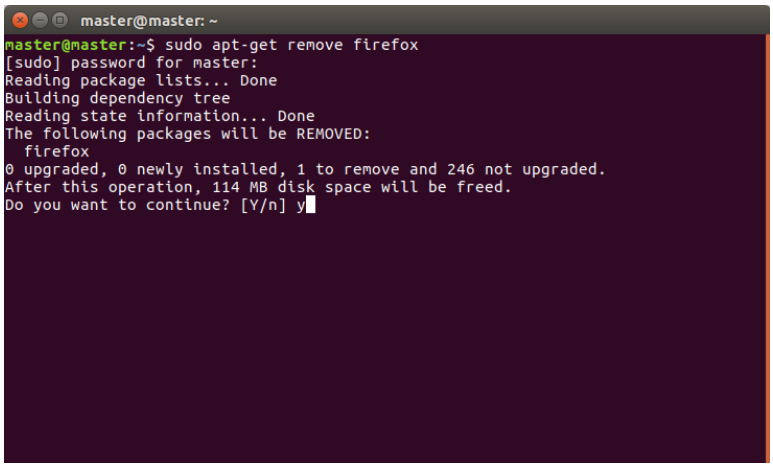
3. Remove

Command synopsis:
```
$ sudo apt-get remove <package_name>
```

Example:

```
$ sudo apt-get remove firefox
```

Output:



Explanation:

The *remove* command uninstalls software from your system. The remove command uninstalls only a single package, it does not uninstalls the package dependencies came with it. After performing this command, you can no longer run the software you removed.

4. Upgrade

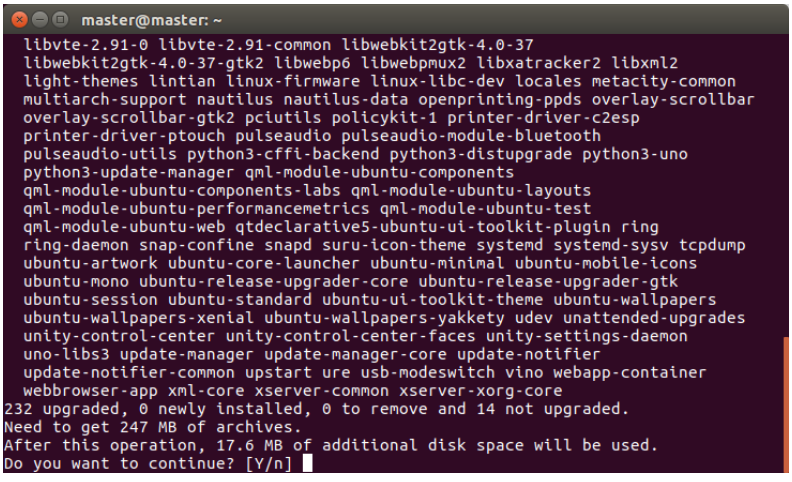
Command synopsis:

```
$ sudo apt-get upgrade
```

Example:

```
$ sudo apt-get upgrade
```

Output:



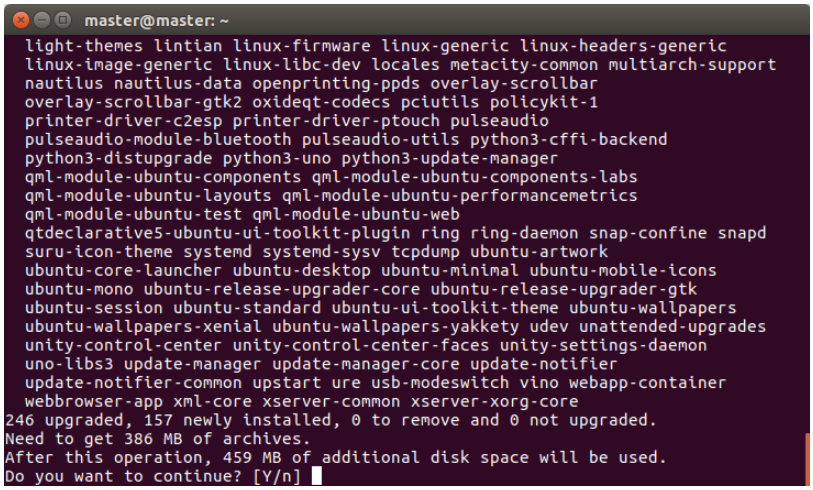
Explanation:
The *upgrade* command downloads and installs all new version of all packages installed in your system. This command depends on your **sources.list** settings, so APT can find new version of packages only if the repository you used does provide them. Remember that *upgrade* is completely different to *update:* upgrade installs packages while update does not install any.

5. Dist-Upgrade

Command synopsis:
```
$ sudo apt-get dist-upgrade
```

Example:
```
$ sudo apt-get dist-upgrade
```

Output:



Explanation:
The dist-upgrade command is a companion of upgrade command, it helps the upgrade command with a “smart conflict resolution”. To give you the difference between both commands: upgrade does not delete any package, while dist-upgrade may delete packages in order to resolve conflicts. This dist-upgrade is usually performed by the user once upgrade command finished. The dist-upgrade command does only *packages upgrade*, not *system version upgrade*, so using it will never upgrade Ubuntu 16.04 into 16.10 for example.

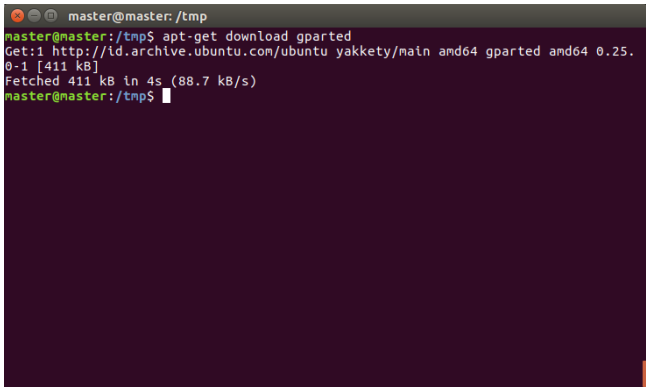
6. Download Only

Command synopsis:
```
$ sudo apt-get download <package_name>
```

Example:

```
$ sudo apt-get download gparted
```

Output:



Explanation:

The *download* command downloads package. This shows that apt-get is basically a download manager. This command will not install the package. The package will be downloaded to the same directory as your command line running. For example, when you perform this command without changing directory, then by default the package stored at your $HOME.

7. Simulate

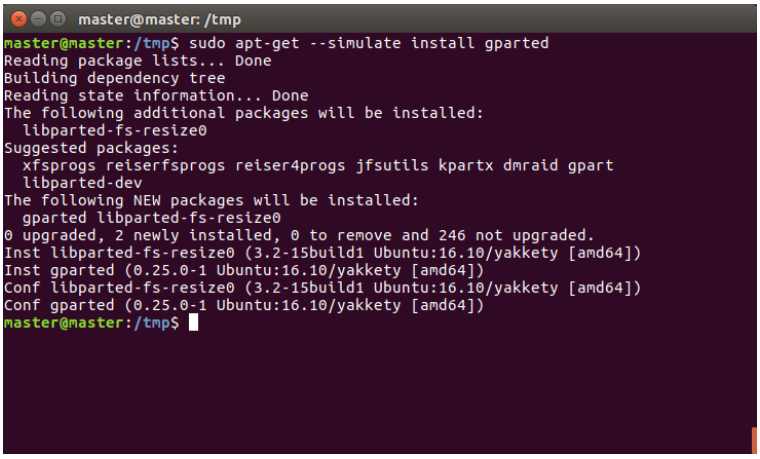
Command synopsis:

```
$ sudo apt-get --simulate install <package_name>
$ sudo apt-get --simulate upgrade
$ sudo apt-get –simulate dist-upgrade
```

Example:

```
$ sudo apt-get --simulate install gparted
```

Output:



Explanation:
This option --simulate is available for apt-get command to simulate (dry run) the package management being processed. This option is usable in the install, remove, upgrade, and dist-upgrade commands. The –simulate is extremely useful to prevent any error before installing/upgrading, because it shows the installing/upgrading process without doing the real installing/upgrading. So you will know if installing certain package there will be an error or not by this --simulate.

8. Add Repository

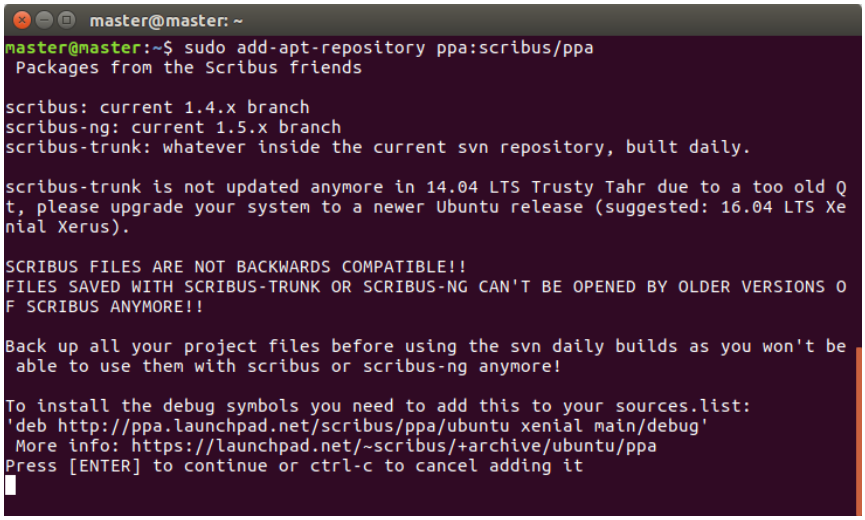
Command synopsis:

```
$ sudo add-apt-repository
```

Example:

```
$ sudo add-apt-repository 'deb http://kambing.ui.ac.id/ubuntu yakkety main
universe'
$ sudo add-apt-repository ‘ppa:gimp/gimp-stable’
```

Output:



Explanation:
This *add-apt-repository* command is not a part of APT, rather, it is a part of software-properties-common package in Ubuntu. This command is a helper to add new repository address to sources.list easily. This command is very common among majority of Ubuntu users, to add new PPA repository (third party repository) to the system so they can install the software available in that repository. To use it, run it as example, and then read the message retrieved, and then press Enter to continue adding the repository address, and after adding you should do Reload to get the map of the new repository.

Note: the reason I put add-apt-repository here is because this command is very famous among PPA users and almost all Ubuntu desktop users use PPA. So for your convenience, I put it here.

9. Print Uris

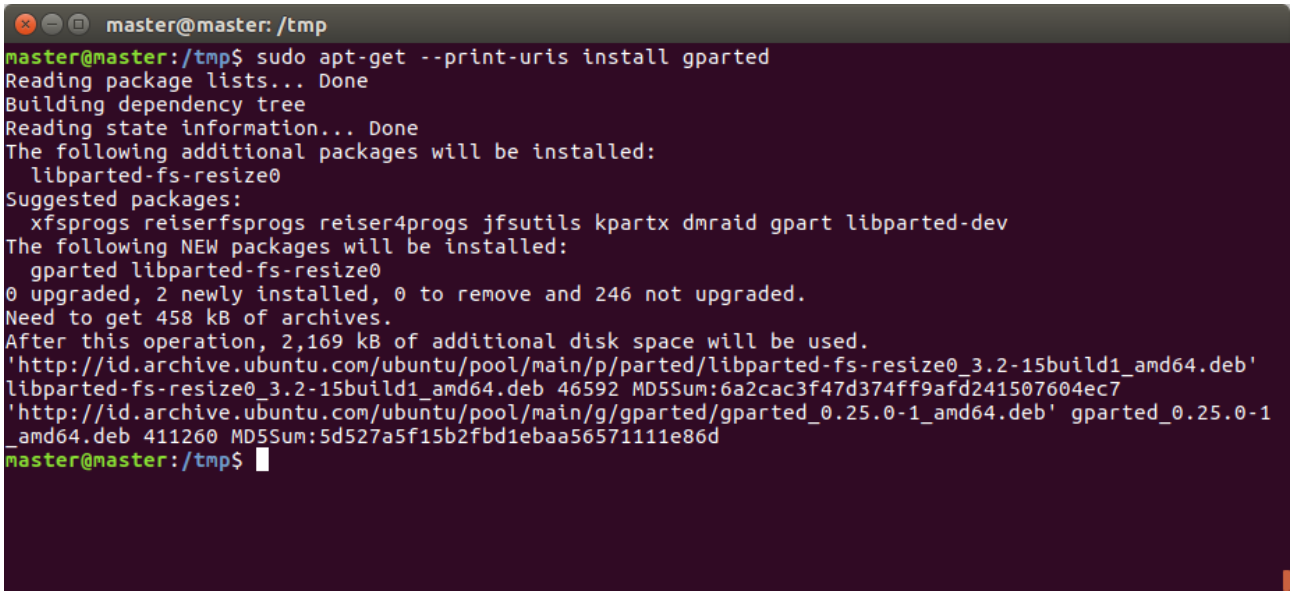
Command synopsis:

```
$ sudo apt-get  --print-uris install <package_name>
```

Example:

```
$ sudo apt-get --print-uris install gparted
```

Output:



Explanation:

This option --print-uris shows all download links of a software package installation. So for example if a gparted installation needs two packages to be downloaded, it shows the two download links of two packages. If you download the two packages manually somewhere and install them by Dpkg, you may install gparted software correctly in an offline Ubuntu system. Yes, this option --print-uris is extremely useful for offline users. This option –print-uris can be used when offline, because APT generates all download links from its “repository map” (local database /var/lib/apt/lists/ and its sources.list). Again, apt-get is basically a download manager so it certainly knows all package download links.

10. Always Yes

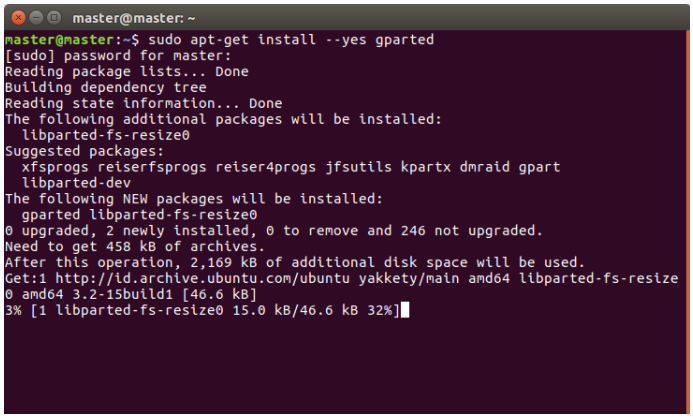
Command synopsis:

```
$ sudo apt-get install --yes <package_name>
```

Example:

```
$ sudo apt-get install --yes gparted
$ sudo apt-get remove --yes gparted
$ sudo apt-get upgrade --yes
$ sudo apt-get dist-upgrade --yes
```

Output:



Explanation:

This option --yes makes the user may leave safely the Terminal while APT is processing. The meaning of --yes here is to answer “y” automatically to APT for all questions possible while doing install/remove/upgrade/dist-upgrade. So, by using this you do not need to answer any further question one by one while doing package management.

Part 3: Basic Sources.list Settings

This beginner's guide Part 3 talks about Ubuntu repository address settings in sources.list. By learning this article you understand what is sources.list and how to setup it according to your needs. In Ubuntu, APT will know what repository to access and what packages to download primarily based on your sources.list settings. So sources.list is very essential for your APT system. This article explains the terms (4 "channels" and 4 "rooms"), gives many practical examples, how to edit, and adds summary and references at the end. I use Ubuntu 16.10 Yakkety Yak for example in this article. And this article is a continuation of Beginner's Guide Part 2 Basic Apt-get. I hope this part is easy for everyone to understand.

1. Repository

What is repository? A repository is a place storing software packages dedicated for a GNU/Linux operating system. Repository is created by the developer of a GNU/Linux operating system, by transforming software from source code into binary, producing a “package” file (.deb) for each software. Repository is located in internet as a server, it allows anyone to download packages. Ubuntu users download software from Ubuntu repository. That is how it works.

To give you more precise sense about Ubuntu repository, notice these facts: every GNU/Linux distribution **has its own** repository, a repository **composed from ten of thousands** packages stored in structured directories, Ubuntu repository is divided in 4 channels and each channels divided in 4 rooms, and a repository provides **binary** and also **source code** version of each software.

2. Ubuntu Repository “Rooms”

Ubuntu differs from Debian. Debian divides its repository in 3 rooms, while Ubuntu divides its repository in 4 rooms. Debian rooms are 3: main, contrib, and nonfree. Ubuntu rooms are 4: main, restricted, multiverse, universe. These divisions are based on their own policies. What you need to know here is the 4 rooms of Ubuntu repository. To give you the (too-simplified) descriptions of them:

- **main**: contains free software, maintained directly by Canonical.
- **restricted**: contains nonfree software, maintained directly by Canonical.
- **universe**: contains free software, maintained by community.
- **multiverse**: contains nonfree software, maintained by community.

Note: please notice that this term "room" is from me, this term is unofficial.

3. Ubuntu Repository “Channels”

Ubuntu repository composed from 4 channels, and every channel has 4 rooms mentioned above. Each channel represents how new the software versions inside are. The user is free to choose what channel enabled and also what room inside channel enabled. The 4 channels of Ubuntu repository are:

- **$release**: the standard channel, you can live in Ubuntu by only using this repo.
- **$release-updates**: the software update channel, use this to get new version of a software.
- **$release-security**: the security updates & fixes channel, use this to get security fixes quickly,
- **$release-backports**: the new version from future Ubuntu version, use this to get the far more new version of a software or even new software not existed in previous Ubuntu repo.

Note: please notice that this term "channel" is also from me, this term is unofficial.

4. Ubuntu Codenames

Ubuntu has *codename* to name each of its releases. Each codename has its own repository. So if Ubuntu has 3 active releases, it means Ubuntu has 3 different repositories. It is important because the user can not use another version’s repository and ultimately because this codename used in sources.list. Here is a list of Ubuntu release codenames starting from 2012 version Ubuntu 12.04:

- 12.04 = precise
- 12.10 = quantal
- 13.04 = raring
- 13.10 = saucy
- 14.04 = trusty
- 14.10 = utopic
- 15.04 = vivid
- 15.10 = wily
- 16.04 = xenial
- 16.10 = yakkety
- 17.04 = zapty

5. Mirror

What is mirror? A mirror is a copy of repository located somewhere else in the internet. So there is an *official* repository, and there are *mirrors* of official repository; they all are containing exactly the same software packages. A mirror is created typically by some third-parties (individuals, or universities, or companies) willing to help the official GNU/Linux repository available more widely. By providing a mirror, it means that users don't need to download software packages directly from the official repository, but instead they can download from a local mirror repository in their country.

A mirror created helps the official repository server. It is clearly a form of cooperation. It is because the amount of download connections decreased at official server when many users switch to another mirror.

Being able to edit sources.list settings means able to use local mirror.

6. Sources.list

What is sources.list? A sources.list is a file containing **list of repository addresses**. APT knows all available packages and where to download them based on settings inside a sources.list. A sources.list file is primarily important for APT and a whole Ubuntu system to install/upgrade software. This file is located at **/etc/apt/sources.list**.

7. Sources.list.d/

What is sources.list.d/? The sources.list.d/ directory is a system directory containing another secondary sources.list files that store additional or third-party repository addresses. This directory is located at **/etc/apt/sources.list.d/** in Ubuntu system.

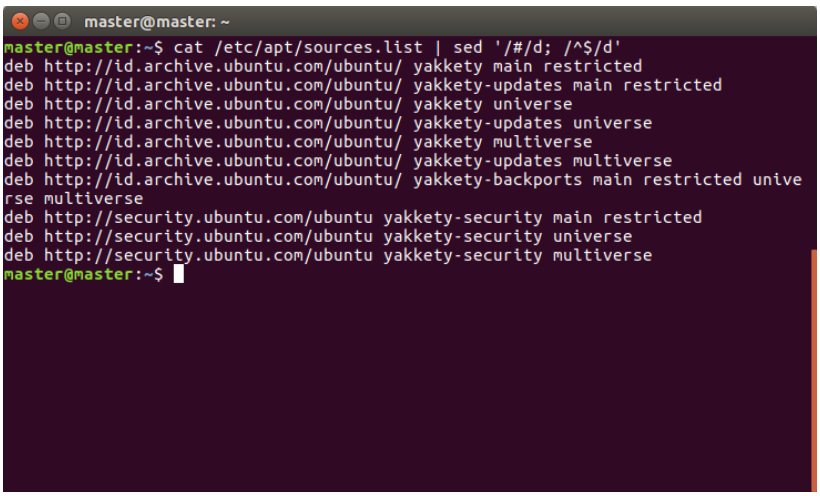
8. How To Read

There are two ways to read sources.list: one with Terminal, one with GUI.

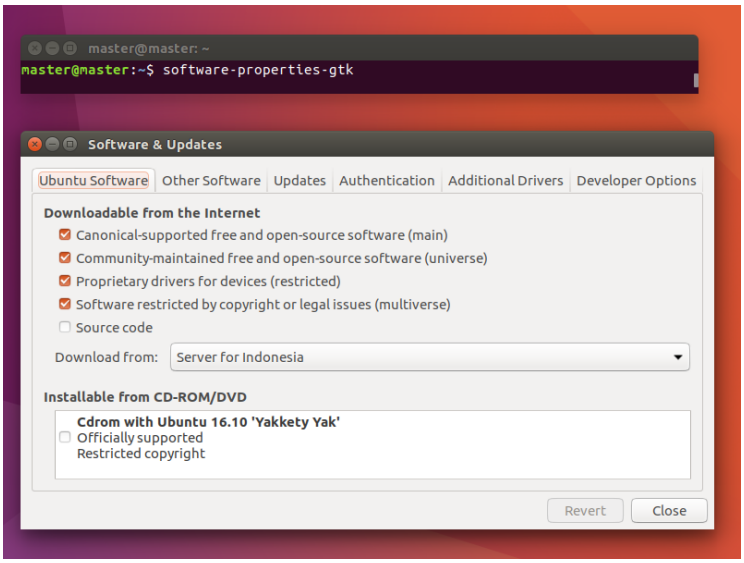
Terminal way: to read a sources.list with Terminal, perform this command line

```
$ cat /etc/apt/sources.list | sed '/#/d; /^$/d'
```

and you should see the sources.list content clearly (without comment lines):



GUI way: open the program Software & Updates from Ubuntu menu or run the command line $ software-properties-gtk. You will see the exactly same sources.list settings but in nice GUI options.



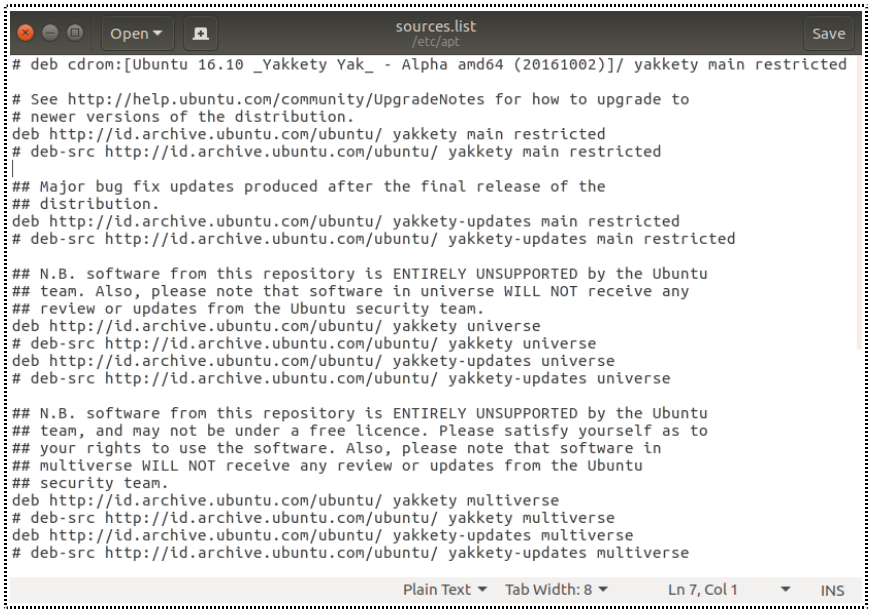
9. How To Edit

To edit a sources.list file, you always need root privilege. You can edit sources.list with any text editor program you like. For example, here you may see step-by-step editing with Gedit:

1. Run command line **$ sudo gedit /etc/apt/sources.list**
2. While asked, type your password anyway in Terminal
3. A Gedit window opened showing sources.list content
4. Edit some text there
5. Save

6. Exit

When you open sources.list file with a text editor, you will see its default content like this:



Do not get confused. This default sources.list content indeed looks very long and complex. But once you understand the structure (explained below) you won't get confused, then you may safely delete the default content and replace it with your own settings.

10. Sources.list General Structure

This is the generic structure of a sources.list setting. It's like poetry. Pay attention to the "4 rooms" and the "$release code" in every "stanza". The 4 rooms mentioned means they are all enabled, the 4 release channels mentioned means they are all enabled.

```
deb [URL] [$release] main
deb [URL] [$release] restricted
deb [URL] [$release] universe
deb [URL] [$release] multiverse

deb [URL] [$release-updates] main
deb [URL] [$release-updates] restricted
deb [URL] [$release-updates] universe
deb [URL] [$release-updates] multiverse

deb [URL] [$release-security] main
deb [URL] [$release-security] restricted
deb [URL] [$release-security] universe
deb [URL] [$release-security] multiverse

deb [URL] [$release-backports] main
```

```
deb [URL] [$release-backports] restricted
deb [URL] [$release-backports] universe
deb [URL] [$release-backports] multiverse
```

11. Sources.list General Structure, Simplified

The very long generic structure above can be simplified into only 4 lines like this. Pay attention in how 4 rooms written, and how the 4 channels is located. This general structure enabled all channels and all rooms in Ubuntu repository.

```
deb [URL] [$release] main restricted universe multiverse
deb [URL] [$release-updates] main restricted universe multiverse
deb [URL] [$release-security] main restricted universe multiverse
deb [URL] [$release-backports] main restricted universe multiverse
```

12. Sources.list General Example

Now, you should understand how to use that generic structure in a real sources.list. For this purpose, I show you here two examples of sources.list: one for 16.10 yakkety and one for 12.04 precise. Both have all channels and rooms enabled. **Pay attention to how [$release*] code replaced, and how [URL] changed**. This method of setting is applicable to all Ubuntu release versions.

Sources.list for yakkety:

```
deb http://id.archive.ubuntu.com/ubuntu/ yakkety main restricted universe multiverse
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-updates main restricted universe multiverse
deb http://security.ubuntu.com/ubuntu/ yakkety-security main restricted universe multiverse
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-backports main restricted universe multiverse
```

Sources.list for precise:

```
deb http://id.archive.ubuntu.com/ubuntu/ precise main restricted universe multiverse
deb http://id.archive.ubuntu.com/ubuntu/ precise-updates main restricted universe multiverse
deb http://security.ubuntu.com/ubuntu/ precise-security main restricted universe multiverse
deb http://id.archive.ubuntu.com/ubuntu/ precise-backports main restricted universe multiverse
```

13. Disabling Repository Component

To manage sources.list means to enable/disable repository certain component. Every user is free to disable any channel and any room of Ubuntu repository, and vice versa. Then, by changing this sources.list, APT will only download software packages from the repository channels/rooms you

chose. **In other words, APT obeys you based on your sources.list setting**. Here are some examples in disabling some channels or rooms in Ubuntu sources.list:

a) Disabling all channels except standard channel:

```
deb http://id.archive.ubuntu.com/ubuntu/ yakkety main restricted universe multiverse
#deb http://id.archive.ubuntu.com/ubuntu/ yakkety-updates main restricted universe multiverse
#deb http://security.ubuntu.com/ubuntu/ yakkety-security main restricted universe multiverse
#deb http://id.archive.ubuntu.com/ubuntu/ yakkety-backports main restricted universe multiverse
```

b) Disabling only *backports* channel:

```
deb http://id.archive.ubuntu.com/ubuntu/ yakkety main restricted universe multiverse
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-updates main restricted universe multiverse
deb http://security.ubuntu.com/ubuntu/ yakkety-security main restricted universe multiverse
#deb http://id.archive.ubuntu.com/ubuntu/ yakkety-backports main restricted universe multiverse
```

c) Disabling all nonfree rooms:

```
deb http://id.archive.ubuntu.com/ubuntu/ yakkety main universe
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-updates main universe
deb http://security.ubuntu.com/ubuntu/ yakkety-security main universe
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-backports main universe
```

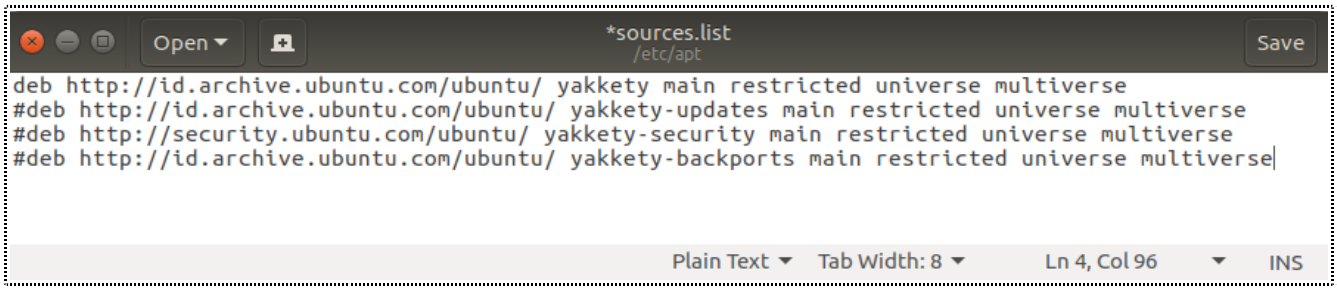
d) Disabling all rooms except *main* room:

```
deb http://id.archive.ubuntu.com/ubuntu/ yakkety main
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-updates main
deb http://security.ubuntu.com/ubuntu/ yakkety-security main
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-backports main
```

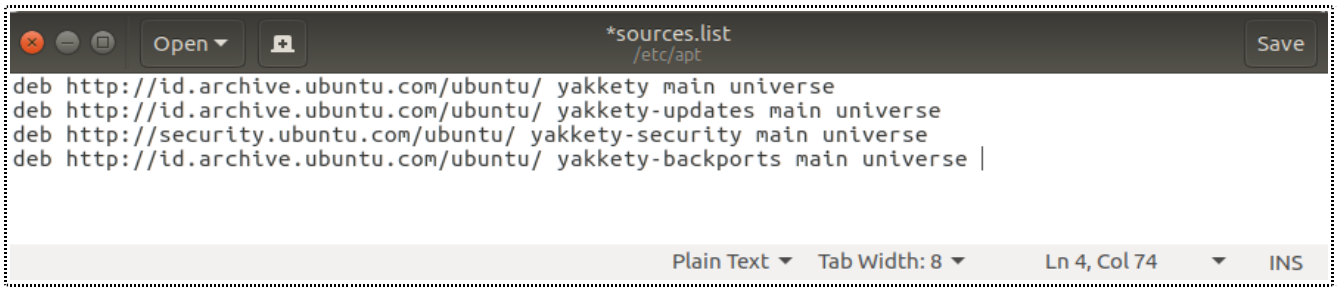
14. How It Looks

Because this article is for beginners, here I show you how it looks when sources.list changed using (a) and (c) examples above. I use Gedit as text editor here.

a) Disabling all channels except standard channel:



c) Disabling all nonfree rooms:



15. Source Code Repository

Ubuntu GNU/Linux distribution distributes software in binary form and also source code form. This means Ubuntu has binary repo and also source code repo at the same time for each release. What I mentioned above is all about binary repo, they are composed of **.deb** packages. So I mention here about source code repo, the repo composed from **.tar.gz** packages. Every Ubuntu user receives the right to access source code of every part of Ubuntu GNU/Linux distribution completely. So whenever you need or you wanna try source code, you can enable the source code repo. This section give you the basic understanding about it.

This is **binary repo** address setting:

```
deb http://id.archive.ubuntu.com/ubuntu/ yakkety main restricted universe
multiverse
```

This is **source code repo** address setting:

```
deb-src http://id.archive.ubuntu.com/ubuntu/ yakkety main restricted universe
multiverse
```

Notice the only difference now is the deb-src code at the beginning of line. The deb code represents binary repo, while you use it then you enable the binary repo. And the deb-src code represents source code repo (source = src), while you use it then you enable the source code repo.

16. Enabling Source Code Repository

To enable any source code repository is just exactly the same as enabling any binary repository previously. You only need to change deb into deb-src. Here is the example for complete 4x4 channelsxrooms repositories enabled both for binary and source:

```
deb http://id.archive.ubuntu.com/ubuntu/ yakkety main restricted universe multiverse
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-updates main restricted universe multiverse
deb http://security.ubuntu.com/ubuntu/ yakkety-security main restricted universe multiverse
deb http://id.archive.ubuntu.com/ubuntu/ yakkety-backports main restricted universe multiverse

deb-src http://id.archive.ubuntu.com/ubuntu/ yakkety main restricted universe multiverse
deb-src http://id.archive.ubuntu.com/ubuntu/ yakkety-updates main restricted universe multiverse
deb-src http://security.ubuntu.com/ubuntu/ yakkety-security main restricted universe multiverse
deb-src http://id.archive.ubuntu.com/ubuntu/ yakkety-backports main restricted universe multiverse
```

Pay attention at two stanzas above: they are both the same except deb-src is for source code. Those two stanzas enable both binary and source code repositories, enable 4 rooms in every of 4 channels; in other words, this setting is the most complete all-repositories-enabled possible for Ubuntu.

To disable any of them, just use the same disabling method already explained.

17. Summary

Here is a summary of this whole **Part 3 Sources.list Settings** article.

1. Sources.list is repository address setting for APT.
2. Once a sources.list changed, apt-get update should be invoked.
3. To disable an address, put ‘#’ character on the beginning of its line.
4. To re-enable an address, remove ‘#’ character on the beginning of line.
5. Ubuntu sources.list composed from 4 channels with 4 rooms each.
6. Ubuntu sources.list composed from binary (deb) and source code (deb-src) repositories.
7. Ordinary users generally just need binary repositories (deb) for 4 channel x 4 rooms.

References

- https://wiki.ubuntu.com/Releases
- https://help.ubuntu.com/community/Repositories/Ubuntu
- https://help.ubuntu.com/community/Repositories
- https://help.ubuntu.com/community/Repositories/CommandLine

- http://askubuntu.com/questions/58364/whats-the-difference-between-multiverse-universe-restricted-and-main
- http://www.howtogeek.com/194247/whats-the-difference-between-main-restricted-universe-and-multiverse-on-ubuntu/

Part 4: PPA & Third-Party Repository

This beginner's guide Part 4 explains **PPA and Third-Party Repository** in Ubuntu. For your information, Ubuntu community is very huge, so there are many third-party repos available providing many software packages not available in Ubuntu official repos. There are 2 PPA examples to be used here, for GIMP and Inkscape, to show how to add the addresses and install the software packages. The aim of this Part 4 is for you to be able to add any PPA address and install software packages from PPA. This guide is a continuation of Part 3 Sources.list. I hope this part helpful for everyone.

1. What Is Repository?

Read the previous part about Sources.list.

2. What Is Third-Party Repository?

Third-party repository is a term to mention “unofficial repository”. It means the software packages stored inside this repo are not available in the official repo; or the repo is created by any party outside the official Ubuntu Developer Team.

3. What Is PPA?

Personal Package Archive (PPA) is a kind of third-party repository to store software packages for Ubuntu users. PPA is a part of Launchpad, thus a service of Canonical, the company behind Ubuntu. The main purpose of a PPA is to deliver software directly to the users without entering the official repository before. **Every PPA has an address** and this address can be added to sources.list in your Ubuntu system. APT will read the address and install the software packages from PPA for you (so again, a PPA is just a repository). This term *PPA* for Ubuntu is more or less synonymous with *AUR* for Arch Linux or *COPR* for Fedora or *SBO* for Slackware. Read more about PPA in Launchpad Help https://help.launchpad.net/Packaging/PPA.

4. Do You Need PPA?

Yes, you will need PPA for many popular software applications that are not available in Ubuntu official repository. There are two possible reasons you need PPA: either it is *software version*, or

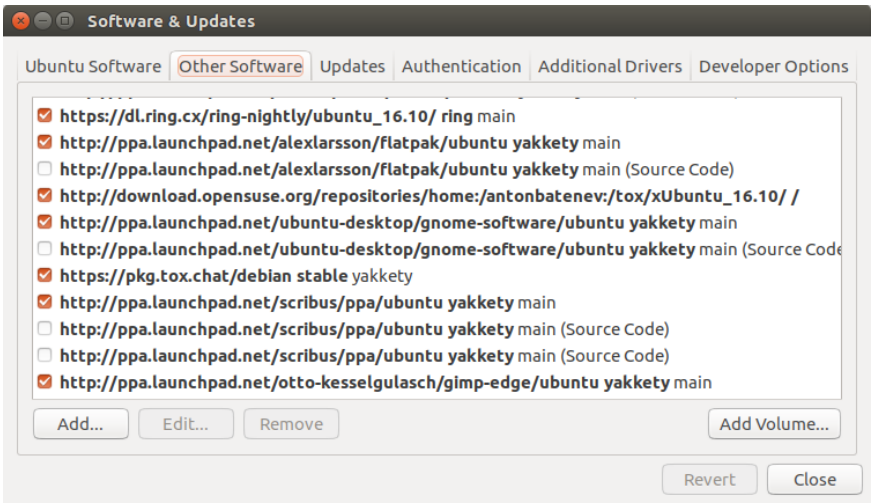
software availability. First condition: sometimes certain software available in *both* PPA and official repo (with the PPA one's are newer, so it's software version); second condition: a lot of certain software available *only* in PPA and *not* in official repo (it's software availability).

For example, there are PPA providing very latest version of *Scribus*, *GIMP*, *Inkscape*, *Blender*, *Krita*, *MyPaint*, *FreeCAD*, and so on. Many users install those software from PPA instead because the official repo versions are considerably old.

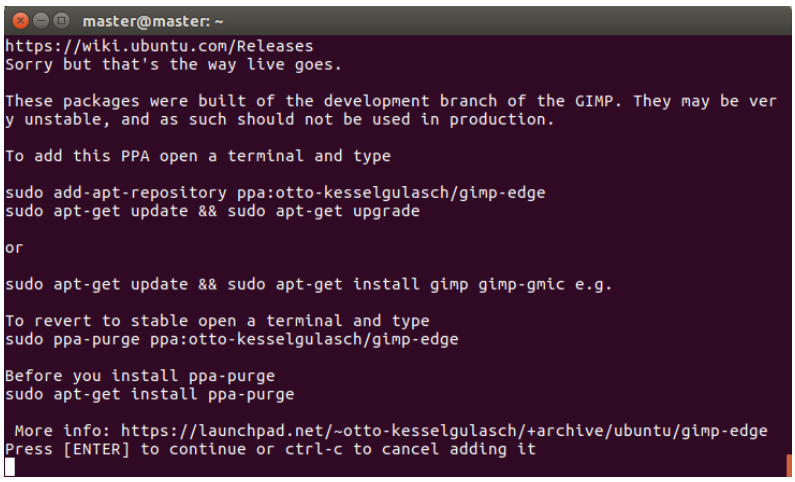
5. Maintain PPA Addresses

To maintain PPA addresses in an Ubuntu system, every Ubuntu user has two choices: using GUI and using console. You can use the GUI program named "Software & Updates" in your menu, or run the command `$ software-properties-gtk` . The rest of this article does not explain the GUI method, instead, it explains the console method.

Using GUI:



Using console:



6. Find A PPA

Look at Launchpad (https://launchpad.net) or search through search engines. The majority of PPA for Ubuntu are available in Launchpad. If you don't have time searching, I have collected many PPA addresses for popular software applications for Ubuntu:

- http://www.ubuntubuzz.com/2016/11/list-of-ppa-repositories-for-ubuntu-16.04-xenial-xerus.html
- http://www.ubuntubuzz.com/2016/11/list-of-ppa-repositories-for-ubuntu-16.10-yakkety-yak.html

7. Add A PPA Address as Sources.list

Generally, to add a PPA address in Ubuntu system, you **edit** the sources.list. Then you should run **apt-get update** command. This apt-get update is needed in order to download the "repository map" of that PPA repository so APT can download packages from it. This article uses 2 PPAs as example, GIMP PPA and Inkscape PPA, to show how to add a PPA address. Both PPA provides software packages for Ubuntu 14.04, 16.04, until 16.10.

- **GIMP PPA** by Otto https://launchpad.net/~otto-kesselgulasch/+archive/ubuntu/gimp-edge
- **Inkscape PPA** by Inkscape Developers Team https://launchpad.net/~inkscape.dev/+archive/ubuntu/stable

8. Add PPA Manually

Edit **/etc/apt/sources.list** file manually by placing the PPA address correctly. I use Gedit text editor again to demonstrate it. See following steps:

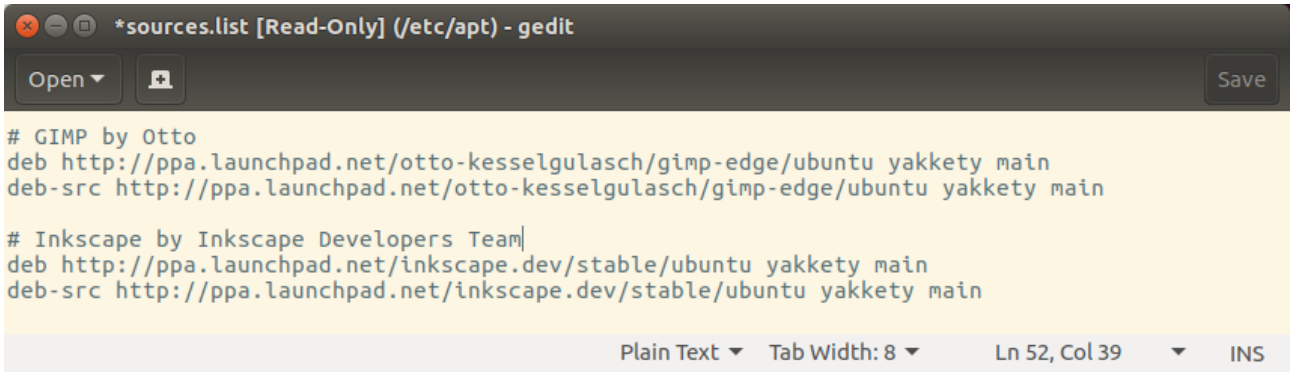
For GIMP PPA:

1. Visit the PPA web page address https://launchpad.net/~otto-kesselgulasch/+archive/ubuntu/gimp
2. Look for "**Technical details about this PPA**" link, click that link, you must see a box appears showing two lines of sources.list.
3. Click "**Choose your Ubuntu version button**" and select your Ubuntu version from there. This will change the two lines automatically to suit your selected Ubuntu version.
4. Copy the two lines of source code there.
5. $ sudo gedit /etc/apt/sources.list
6. Paste the two lines you copied into a new blank line at the most bottom.
7. Save the sources.list file.
8. $ sudo apt-get update

For Inkscape PPA:

1. Visit the PPA web page address https://launchpad.net/~inkscape.dev/+archive/ubuntu/stable.
2. Do point 2 until 8 like the GIMP PPA above.

Screenshot:



9. Add PPA via Special Command

The most common and easiest way to add PPA address is using **add-apt-repository** command line. This command will automatically add a special .list file to **/etc/apt/sources.list.d/** directory for each PPA. In other words, this command doesn't touch the main sources.list to add PPA. To demonstrate it, I show here the 2 examples again.

For GIMP PPA:

1. Visit the PPA web page address https://launchpad.net/~otto-kesselgulasch/+archive/ubuntu/gimp-edge.
2. Look for one address code “ppa:packager_name/ppa_name” and for this case it is **ppa:otto-kesselgulasch/gimp-edge.**
3. Copy that PPA address code.
4. $ sudo add-apt-repository **ppa:otto-kesselgulasch/gimp-edge**
5. Terminal will ask for your permission. Press Enter to go next, or press Ctrl+C to cancel.
6. $ sudo apt-get update

For Inkscape PPA:

1. Visit the PPA web page address https://launchpad.net/~inkscape.dev/+archive/ubuntu/stable.
2. Look for one address code “ppa:packager_name/ppa_name” and for this case it is **ppa:inkscape.dev/stable**.
3. Do point 3 until 6 like GIMP PPA above.

10. Install Software from PPA

This is the purpose of using a PPA repository. To install software package from PPA, simply run **apt-get install** command with the <package_name> of the software available in that PPA. Regarding the 2 examples mentioned, so the command lines are:

Synopsis:

```
$ sudo apt-get install <package_name>
```

GIMP from PPA:

```
$ sudo apt-get install gimp
```

Inkscape from PPA:

```
$ sudo apt-get install inkscape
```

11. Comparing Package Versions

Once you add a PPA address and obtain its “repository map”, you can see the both version of same software packages both in PPA and official repo. Use **apt-cache** command to do it like this.

Synopsis:

```
$ apt-cache policy <package_name>
```

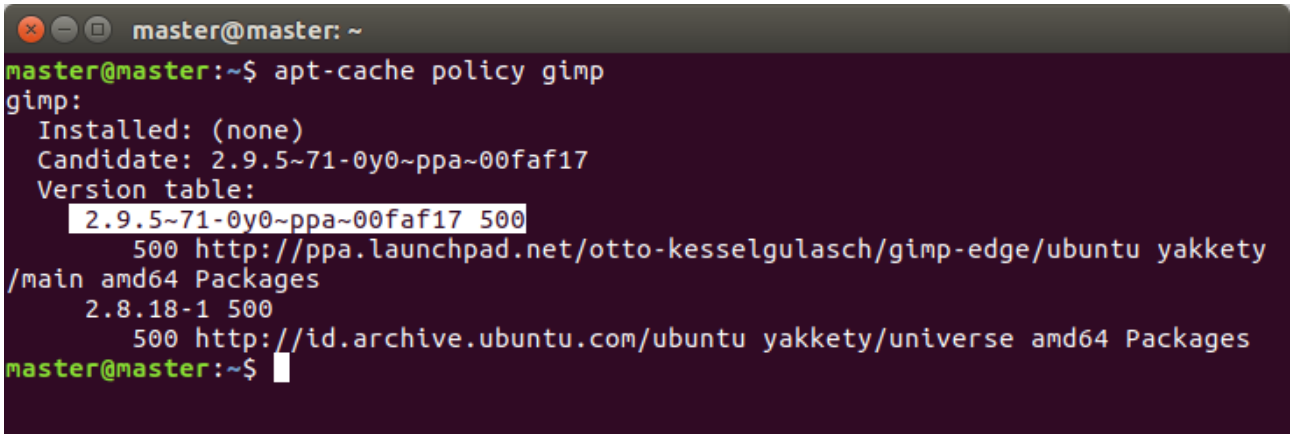
GIMP versions:

```
$ apt-cache policy gimp
```

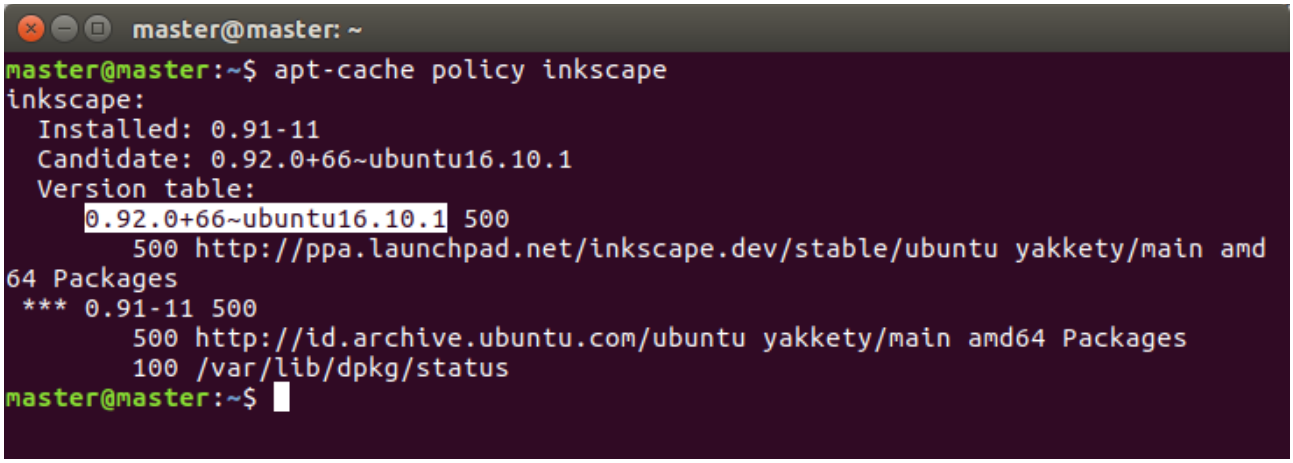
Inkscape version:

```
$ apt-cache policy inkscape
```

Output for GIMP:



Output for Inkscape:



Explanation:

- For GIMP: you see that there are two versions, version 2.9.5 from PPA and version 2.8.18 from official repo.
- For Inkscape: you see that there are two versions, version 0.92 from PPA and version 0.91 from official repo.

12. Remove PPA Address Automatically

This method is easier for beginner. To remove a PPA address (only the address, not the software) from your sources.list system, use **add-apt-repository** command again. Remember that you need to run apt-get update once you change any sources.list setting. I show you for the 2 examples above.

For GIMP:

```
$ sudo add-apt-repository --remove ppa:otto-kesselgulasch/gimp
$ sudo apt-get update
```

For Inkscape:

```
$ sudo add-apt-repository --remove  ppa:inkscape.dev/stable
$ sudo apt-get update
```

Explanation:

The command above will delete the PPA address. However, that command does not delete its *.list file. You may check /etc/apt/sources.list.d/ directory for the file.

13. Remove PPA Address Manually

This method is needed when you experience some trouble. To remove PPA address manually, if you added it *manually* too, then you just need to delete its lines in sources.list. But to remove PPA address manually if you added it *automatically* (via add-apt-repository), then you should know the files and delete them manually.

Method 1:

- $ sudo gedit /etc/apt/sources.list
- Look for your PPA address line there, such as gimp-edge or inkscape-stable.
- Delete that lines.
- Save sources.list file.
- $ sudo apt-get update

Method 2:

- Look at /etc/apt/sources.list.d/ directory.
- Look for any .list file named with your PPA address name, such as gimp-edge or inkscape-stable.
- Assume the PPA address file name is “gimp-edge.list” then the delete command is:

- $ sudo rm /etc/apt/sources.list.d/gimp-edge.list*
- Then perform a reload:
- $ sudo apt-get update

14. Remove Software Packages from PPA

To remove software package installed from PPA, use apt-get remove normally. There are 2 examples:

For GIMP:

```
$ sudo apt-get remove gimp
```

For Inkscape:

```
$ sudo apt-get remove inskcape
```

Note:

Remember that deleting the package is *not* the same as deleting the PPA address. After removing the package, if you install the same package again next time, your system will choose the PPA version not the official repository version. For that reason, you may delete the PPA address either so next time APT will install package from official repository.

Additional Information

Once you use Ubuntu you use free software and live in free software community. One of the meaning of being *free* is everyone in whole community *has the right* to distribute software. So it means there are still many third-party repositories available for Ubuntu outside Launchpad. One of the biggest of them is **openSUSE OpenBuildService** https://build.opensuse.org.

References

- http://askubuntu.com/questions/4983/what-are-ppas-and-how-do-i-use-them
- http://askubuntu.com/questions/307/how-can-ppas-be-removed
- http://unix.stackexchange.com/questions/6766/is-there-is-a-ppa-service-equivalent-in-the-fedora-world