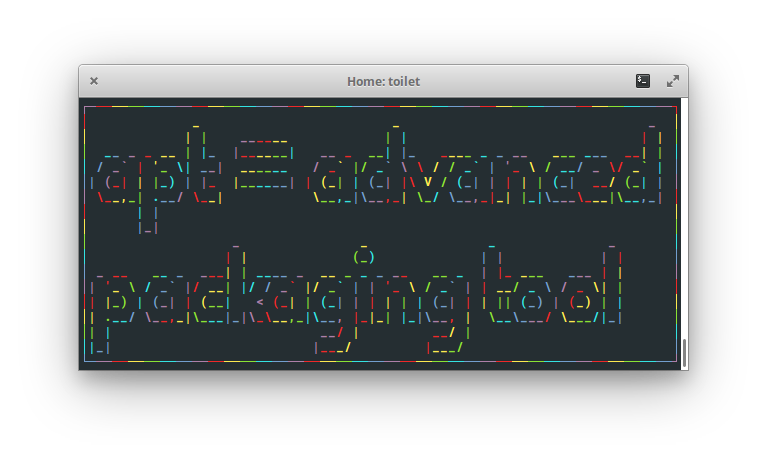
A Clear in-depth Guide to Apt

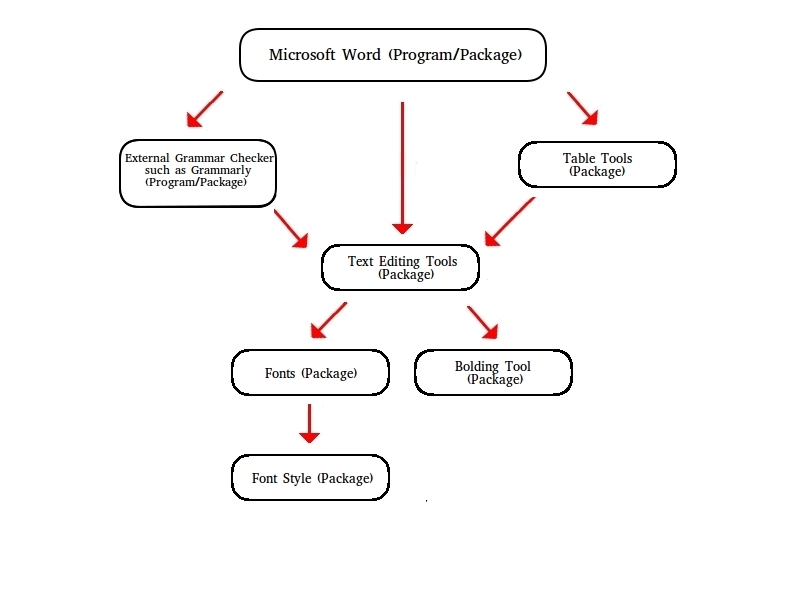
Prerequisites: Basic Terminal Usage Article Type: Informative



Apt, or Advanced Packaging Tool, is a handy command line interface used for managing packages and installing software on Ubuntu-based operating systems. After learning the functions of apt, you will be able to install/uninstall packages, manage installed packages and explore other available packages.

Before we can understand how apt package manager works, we need to know what a package is and how it differs from a program.

A package is a bundle of code designed for one purpose. It could be used to modify a function in other software, or give your operating system a complete new functionality. Thus, a program can be considered a package; it just has to be a bundle of code.

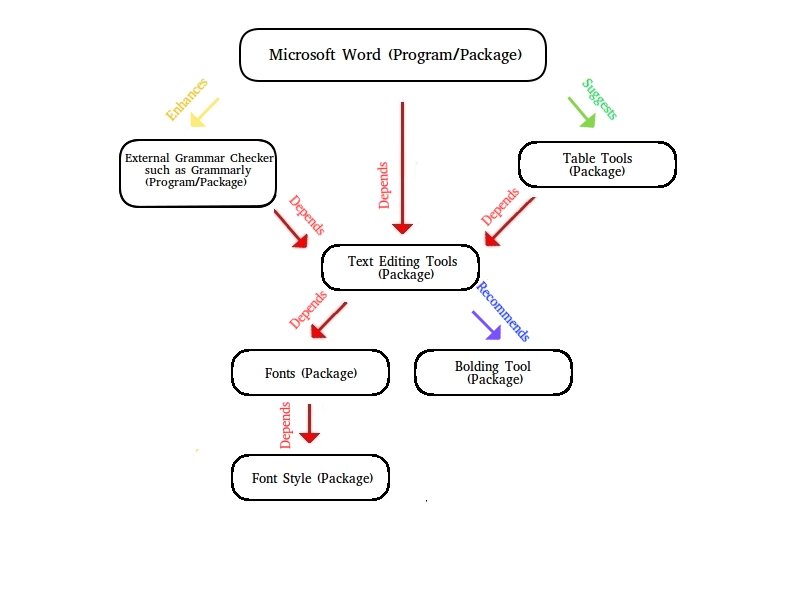
The definition of a package is simple, but the way of using it is slightly complex. Packages are related to one another and cannot be considered as an independent piece of software. As an example, let us consider a program/package such as Microsoft Word. It contains tools like bolding a word, changing a font, and changing font size. Each of these tools can be considered a package or a sub-package of Microsoft Word. We can even consider Windows as a package, which in this case, Microsoft Word would be considered a sub-package. As you can see, what is considered a package can be very broad, but it is important to understand that packages depend on one another to function. The picture below summarizes the structure of a simple package quite well:

A possible implementation of the Microsoft Word package structure

There are four types of sub-packages: depends, recommends, suggests, and enhances, which are briefly explained in the table below. All examples of sub-packages are relative to the Microsoft Word example.

Types of Subpackages Explained

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Depends | Recommends | Suggests | Enhances |
| Definition | A package which the program would not be able to function without | A package that the program highly depends on but does not require it to function | A package that may be useful to the program but it is perfectly fine to run the program without them | A package that can add or enhance a functionality, usually in a way that the program is not originally designed for |
| Examples | Fonts, Save tool | Bold, Underline Tools | Table Tools  Bibliography Tools | External Spell Checker Tool such as Grammarly |



A possible implementation of Microsoft Word’s package structure along with their relationships

When you download a package, you do not have to download the dependencies separately; apt will install the dependencies for the program you desire.

All your packages installed come from a *software repository*, or *repo* for short. You can consider a repo as a cloud, similar to Google Drive. Each repo contains packages that were created for a specific purpose. Several repos come with a fresh-installation of Ubuntu, however, to install more diverse software with apt, you may have to add new repos.

Apt manages its available packages for download in a local database. Each entry in the database references an available package in a linked repo. This helps apt perform its functions faster as the list of packages are cached locally. However, as newer software often gets added to the repos, the database becomes outdated. Thus to work with the latest software, you must update this database regularly. This is called *updating your package manager*. You must also update your package manager after you add a new repo to cache the packages in the new repo as well.

Basic apt commands:

**Installing a package:**

To install a program, you will need to know the package name. For example, use to install firefox. If you attempt to install a package that you have already installed, it will update the package for you instead. You can also change the *install* into *reinstall* to reinstall the given package.

**Installing a *deb* package:**

Sometimes you may come across a deb file when downloading a program. You can also install *deb* packages with *apt install* by passing the downloaded *deb* file location as a parameter of *apt install*.

**Adding a repo:**

You can add a repo with the *add-repository* command. You can usually find the repository name on the program’s webpage.

**Updating package manager:**

This will update apt’s database on the packages available for installation.

**Updating your installed packages:**

Both *apt upgrade* and *apt full-upgrade* will ensure all your packages are up to date. The difference is that *apt full-upgrade* will also delete packages that are required for an update, whereas *apt update* won’t. Apt will only upgrade the packages you installed with apt. If you have installed programs through .tar files, you will have to upgrade them yourself.

**Removing a package:**

*Apt remove* will remove the package without deleting the configuration files. With this way, if you reinstall the package, your settings will remain intact. *Apt purge* will remove the package as well as the configuration files.

**Removing unnecessary packages:**

Some packages are auto-installed as dependencies of another package. When the package that requires these dependencies is uninstalled, the dependencies are not always uninstalled. You can use *autoremove* to delete these packages.

**Searching for packages:**

This will list all packages that match with your search term.

**Listing all installed packages:**

This will only the packages installed with apt.

**Viewing details regarding a package:**

**Learn more about apt:**

Apt configuration files locations:

In general, you do not need to know where the apt configuration files are located. However, they may be good to know should you need to debug some problem with apt.

Apt config file: /etc/apt/apt.conf

Apt linked-repositories list: /etc/apt/sources.list.d/

You may have seen other package managers such as apt-get, pip and dpkg. There exists many package managers, and each of them is unique in their own way. For example, pip is a package manager for python and dpkg is a lower level package manager. Although this article does not directly teach you how the other package managers work, all package managers generally work in a similar manager. Being able to apt means that you easily learn how to use other package managers!

Resources:

If you want to learn more regarding the relationships between packages, check the link below:

<https://www.debian.org/doc/debian-policy/ch-relationships.html>, Debian

For more information on the apt commands, check:

<https://www.howtogeek.com/63997/how-to-install-programs-in-ubuntu-in-the-command-line/>, How to Geek