the following command worked.

sudo rm -rf /var/lib/dpkg/updates/\*

sudo rm -rf /var/lib/apt/lists/\*

sudo rm /var/cache/apt/\*.bin

sudo apt-get clean

sudo apt-get autoremove

sudo apt-get update

sudo dpkg --configure -a

sudo apt-get install -f

if still getting same error then try

sudo dpkg --remove --force-remove-reinstreq package\_name

after removing package update your system and restart once.

Sometimes the package management system is so broken (often in a circular dependency chain) that the only combination of commands I have found to remove problematic packages are:

* cd /var/lib/dpkg/info ; rm <packagename.\*>
* dpkg --purge --force-remove-reinstreq <packagename>
* OR dpkg --remove --force-remove-reinstreq <packagename>
* Then apt-get install <packagename> to re-install the package

Sometimes, the state-override file may also need to be fixed, and the above procedure may throw up errors regarding that .. How to fix the state override files depends on whether it is an error flagging a user or not.

Generally:

* the state override files are in: /var/lib/dpkg/statoverride
* You can update it by running: dpkg-statoverride --remove <stateoverride-filename>
* You can even edit them by hand if you know what you are doing
* **Re: How to fix broken packages?**
* Try
* Code:
* sudo apt-get update
* to update your package list.  
  Then
* Code:
* sudo apt-get autoclean
* to clean up any partial packages.  
  Then
* Code:
* sudo apt-get clean
* to clean up the apt cache.
* Code:
* sudo apt-get autoremove
* will clean up any unneeded dependencies.  
  If while doing this you can identify the broken package this code will very forcefully remove it.
* Code:
* sudo dpkg --remove -force --force-remove-reinstreq package name
* Change package name to the real name of course.

|  |  |  |
| --- | --- | --- |
| Mostly, apt-get does the following things:   * checks for dependencies (and asks to install them), * downloads the package, verifies it and then tells dpkg to install it.   dpkg will:   * extract the package and copy the content to the right location, and check for pre-existing files and modifications on them, * run [package maintainer scripts](https://www.debian.org/doc/debian-policy/ch-maintainerscripts.html): preinst, postinst, (and prerm, postrm before these, if a package is being upgraded) * execute some actions based on [triggers](https://wiki.debian.org/DpkgTriggers)   You might be interested in the maintainer scripts, which are usually located at /var/lib/dpkg/info/<package-name>.{pre,post}{rm,inst}. These are usually shell scripts, but there's no hard rule. For example:  $ ls /var/lib/dpkg/info/xml-core.{pre,post}{rm,inst}  /var/lib/dpkg/info/xml-core.postinst  /var/lib/dpkg/info/xml-core.postrm  /var/lib/dpkg/info/xml-core.preinst  /var/lib/dpkg/info/xml-core.prerm   |  |  | | --- | --- | | [share](http://askubuntu.com/a/540943)[improve this answer](http://askubuntu.com/posts/540943/edit) | answered Oct 24 '14 at 10:30  [[https://www.gravatar.com/avatar/0ce9def9898b5e9e0339ccc26c57ae94?s=32&d=identicon&r=PG](http://askubuntu.com/users/158442/muru)](http://askubuntu.com/users/158442/muru)  [muru](http://askubuntu.com/users/158442/muru) 38.2k85391 | |
|  | add a comment |

|  |  |
| --- | --- |
| up vote 22 down vote | **In short**: apt-get install does everything that is needed that your system can successfully execute the new installed software application.  **Longer:**  **Preliminaries:**  From the [manpage](http://manpages.ubuntu.com/manpages/trusty/en/man8/apt-get.8.html):  All packages required by the package(s) specified for installation will also be retrieved and installed.  Those packages are stored on a repository in the network. So, apt-get downloads all the needed ones into a temporary directory (/var/cache/apt/archives/). They will be downloaded from a web- or a ftp-server. They are specified in the so called sources.list; a list of repositories. From then on they get installed one by one procedurally.  The first ones are the ones, that have no further dependencies; so no other package has to be installed for them. Trough that, other packages (that had dependencies previously) have now no dependencies anymore. The system keeps doing that process over and over until the specified packages are installed.  Each package undergoes an installation procedure.  **Package installation:**  In Debian-based Linux distributions, as Ubuntu, those packages are in a specified standardized format called: deb - [The Debian binary package format](http://en.wikipedia.org/wiki/Deb_%28file_format%29).  Such a package contains the files to be installed on the system. Also they contain a [control file](https://www.debian.org/doc/debian-policy/ch-controlfields.html). That file contains scripts that the packaging system should execute in a specific situation; the so called [maintainer scripts](https://wiki.debian.org/MaintainerScripts). Those scripts are splited in:   * preinst: before the installation of the files into the systems filehierarchy * postinst: after the installation * prerm: before the uninstallation * postrm: after the uninstallation   There is an interessting picture, showing the procedure of an installation of a new package:  installation  There are also more control-files, the most important are as follows:   * control: A [list](https://www.debian.org/doc/debian-policy/ch-controlfields.html#s-sourcecontrolfiles) of the dependencies, and other useful information to indetify the package * conffiles: A [list](https://www.debian.org/doc/manuals/maint-guide/dother.en.html#conffiles) of config files (usually those in /etc) * debian-binary: contains the deb-package version, [currently 2.0](http://tldp.org/HOWTO/Debian-Binary-Package-Building-HOWTO/x60.html#AEN66) * md5sums: A list of md5sums of each file in the package for vertifying * templates: A file with [error descriptions and dialogs](https://www.debian.org/doc/packaging-manuals/debconf_specification.html#AEN45) during installation |

Well, we could kind of say that apt-get is one such “store.” On Linux (and I mean in general), there are two main ways of installing applications.

1. Either you download the *source code* of a particular application (with wget to download an archive from the web, or sometimes with git), and then run
   * ./configure, that checks dependencies and creates a Makefile for your architecture
   * make, which reads the Makefile and compiles the source code into binary files
   * make install, which moves all the files to their proper location on your system.
2. Or you use a **package manager** (like apt-get/aptitude on Debian), a *much* more powerful method.

# [What is the real difference between “apt-get” and “aptitude”?](http://unix.stackexchange.com/questions/767/what-is-the-real-difference-between-apt-get-and-aptitude-how-about-wajig)

The most obvious difference is that aptitude provides a terminal menu interface (much like Synaptic in a terminal), whereas apt-get does not.

Considering only the command-line interfaces of each, they are quite similar, and for the most part, it really doesn't matter which one you use. Recent versions of both will track which packages were manually installed, and which were installed as dependencies (and therefore eligible for automatic removal). In fact, I believe that even more recently, the two tools were updated to actually share the same database of manually vs automatically installed packages, so cases where you install something with apt-get and then aptitude wants to uninstall it are mostly a thing of the past.

There are a few minor differences:

* aptitude will automatically remove eligible packages, whereas apt-get requires a separate command to do so
* The commands for *upgrade* vs. *dist-upgrade* have been renamed in aptitude to the probably more accurate names *safe-upgrade* and *full-upgrade*, respectively.
* aptitude actually performs the functions of not just apt-get, but also some of its companion tools, such as apt-cache and apt-mark.
* aptitude has a slightly different query syntax for searching (compared to apt-cache)
* aptitude has the *why* and *why-not* commands to tell you which *manually installed* packages are preventing an action that you might want to take.
* If the actions (installing, removing, updating packages) that you want to take cause conflicts, aptitude can suggest several potential resolutions. apt-get will just say "I'm sorry Dave, I can't allow you to do that."

There are other small differences, but those are the most important ones that I can think of.

In short, aptitude more properly belongs in the category with Synaptic and other higher-level package manager frontends. It just happens to also have a command-line interface that resembles apt-get.

|  |  |
| --- | --- |
| 214 down vote accepted | In a nutshell, apt-get update doesn't actually install new versions of software.   * apt-get update downloads the package lists from the repositories and "updates" them to get information on the newest versions of packages and their dependencies. It will do this for all repositories and PPAs. From <http://linux.die.net/man/8/apt-get>:   Used to re-synchronize the package index files from their sources. The indexes of available packages are fetched from the location(s) specified in /etc/apt/sources.list(5). An update should always be performed before an upgrade or dist-upgrade.   * apt-get upgrade will fetch new versions of packages existing on the machine if APT knows about these new versions by way of apt-get update.   From <http://linux.die.net/man/8/apt-get>:  Used to install the newest versions of all packages currently installed on the system from the sources enumerated in /etc/apt/sources.list(5). Packages currently installed with new versions available are retrieved and upgraded; under no circumstances are currently installed packages removed, nor are packages that are not already installed retrieved and installed. **New versions of currently installed packages that cannot be upgraded without changing the install status of another package will be left at their current version.** [Emphasis mine] An update must be performed first so that apt-get knows that new versions of packages are available.   * apt-get dist-upgrade will do the same job which is done by apt-get upgrade, plus it will also intelligently handle the dependencies, so it might remove obsolete packages or add new ones. See here: [What is "dist-upgrade" and why does it upgrade more than "upgrade"?](http://askubuntu.com/questions/81585/what-is-dist-upgrade-and-why-does-it-upgrade-more-than-upgrade)   From <http://linux.die.net/man/8/apt-get>:  In addition to performing the function of upgrade, this option also intelligently handles changing dependencies with new versions of packages; apt-get has a "smart" conflict resolution system, and it will attempt to upgrade the most important packages at the expense of less important ones, if necessary. The /etc/apt/sources.list(5) file contains a list of locations from which to retrieve desired package files. See also apt\_preferences(5) for a mechanism for over-riding the general settings for individual packages.  You can combine commands with && as follows:  sudo apt-get update&&sudo apt-get install foo bar baz foo-dev foo-dbg  or to get newest versions possible as per version requirements of dependencies:  sudo apt-get update&&sudo apt-get dist-upgrade  You need sudo both times, but since sudo by default doesn't prompt you within 5 or so minutes since the last sudo operation, you will be prompted for your password only once (or not at all). |

**dpkg** only installs a package, so doing dpkg -i packageName.deb will only install this Deb package, and will notify you of any dependencies that need to be installed, but it will ***not*** install them, and it will ***not*** configure the packageName.deb because well...the dependencies are not there.

**apt-get** is a [**Package Management System**](http://en.wikipedia.org/wiki/Package_management_system) that handles the installation of Deb packages on [Debian-based Linux distributions](http://en.wikipedia.org/wiki/List_of_Linux_distributions#Debian-based). A Package Management System is a set of tools that will help you install, remove, and change packages easily. So apt-get is like a clever dpkg.

**aptitude** then came along. It uses the [libraries apt-get uses](https://debian-handbook.info/browse/stable/sect.apt-frontends.html) and actually has an interactive UI (user interface). If you want to see this UI, simply type aptitude in the terminal. That's aptitude. It leverages the libraries to provide more options and perks than apt-get. For example, aptitude will automatically remove eligible packages, while apt-get needs a separate command to do so. But, in the end, doing sudo aptitude install packageName.deb should at least be the same as sudo apt-get install packageName.deb. There might be subtle differences here and there that I do not know about, but they will both look for the dependencies and do all that stuff. You can read [the answer here](http://unix.stackexchange.com/q/767/38128) for more information on the differences between aptitude and apt-get.

**gdebi** is another tool that is kind of a mixture between apt-get and aptitude. When you use it to install a .deb package (gdebi packageName.deb), it will identify the missing dependencies, install them using apt-get, and then finally install and configure the package using dpkg. It even has a [simple and neat GUI that gives you information](http://i.imgur.com/lcfsv1u.png) about the .deb package, the files included in the package, and what dependencies need to be installed. To see this GUI, you would do gdebi-gtk packageName.deb. You can give gdebi a try by installing it with sudo apt-get install gdebi

# apt-get

Package management via apt-get runs hand-in-hand with the /etc/apt/sources.list file.

apt-get install <package\_name> installs a new package onto your computer.

apt-get build-dep <package\_name> This command searches the repositories and installs the build dependencies for . If the package is not in the repositories it will return an error.

apt-get install <package1\_name> <package2\_name> <package3\_name> apt-get allows multiple package installation. separate the packages with a space.

auto-apt run <command\_string> When invoked, the auto-apt command automatically installs packages upon missing file access. If a program tries to access a file known to belong in an uninstalled package, auto-apt will install that package using apt-get. This feature requires apt and sudo to work.

# aptitude

aptitude provides the functionality of dselect and apt-get as well as many additional features not found in either program.

aptitude has a shell of its own which is keyboard or mouse activated and runs in a terminal window

aptitude build-dep <package> - Install the build-dependencies of packages - which means the packages needed to compile (or build) its source package. For example, many packages need debhelper or autotools to be built, but don't need them to run.

# dpkg

is a tool to install, build, remove and manage Debian packages. [link](http://manpages.ubuntu.com/manpages/intrepid/man1/dpkg.1.html)

dpkg -i <package.deb> installs a Debian package onto your computer. It does not install any dependencies as far as I can find out from the help files.

For details type dpkg --help into a terminal window.

# APT and Dpkg Quick Reference Sheet

**Matthew Danish**

### Common APT usage

**apt-get install <package>** Downloads <package> and all of its dependencies, and installs or upgrades them. This will also take a package off of *hold* if it was put on. See below for more info on *hold.*  
  
**apt-get remove [--purge] <package>** Removes <package> and any packages that depend on it. --purge specifies that packages should be *purged*, see dpkg -P for more information.  
  
**apt-get update** Updates packages listings from Debian mirrors, should be run at least once a day if you install anything that day, and every time after /etc/apt/sources.list is changed.  
  
**apt-get upgrade [-u]** Upgrades all packages installed to newest versions available. Will not install new or remove old packages. If a package changes dependencies and requires installation of a new package, it will not be upgraded, it will be put on *hold* instead. apt-get upgrade will not upgrade packages put on *hold* (that is the meaning of *hold*). See below for how to manually put packages on *hold*. I suggest the `-u' option as well, because then you can see what packages are going to be upgraded.  
  
**apt-get dist-upgrade [-u]** Similar to apt-get upgrade, except that *dist-upgrade* will install or remove packages to satisfy dependencies.  
  
**apt-cache search <pattern>** Searches packages and descriptions for <pattern>.  
  
**apt-cache show <package>** Shows the full description of <package>.  
  
**apt-cache showpkg <package>** Shows a lot more detail about <package>, and its relationships to other packages.  
  
**dselect   
console-apt   
aptitude   
gnome-apt** Graphical front ends to *APT* (some of these may be in their own package, that must be installed before use). While dselect is arguably the most powerful, it's also the oldest and hardest to use.

### Common Dpkg usage

**dpkg -i <package.deb>** Installs a Debian package file; one that you downloaded manually, for example.  
  
**dpkg -c <package.deb>** Lists the contents of <package.deb>, a .deb file.  
  
**dpkg -I <package.deb>** Extracts package information from <package.deb>, a .deb file.  
  
**dpkg -r <package>** Removes an installed package named <package>  
  
**dpkg -P <package>** Purges an installed package named <package>. The difference between *remove* and *purge* is that while *remove* only deletes data and executables, *purge* also deletes all configuration files in addition.  
  
**dpkg -L <package>** Gives a listing of all the files installed by <package>. See also dpkg -c for checking the contents of a .deb file.  
  
**dpkg -s <package>** Shows information on the installed package <package>. See also apt-cache show for viewing package information in the Debian archive and dpkg -I for viewing package information extracted from a .deb file.  
  
**dpkg-reconfigure <package>** Reconfigures an installed package, if it uses *debconf* (*debconf* provides that consistent configuration interface for package installation). You can reconfigure *debconf* itself if you want to change the front-end or priority of questions asked. For example, to reconfigure *debconf* with the dialog front-end, you simply run:

dpkg-reconfigure --frontend=dialog debconf

**echo ``<package> hold'' | dpkg --set-selections** Put <package> on *hold* (command line method)  
  
**dpkg --get-selections ``<package>''** Get the current status of <package> (command line method)  
  
**dpkg -S <file>** Searches for <file> in package database, telling you which packages have that file in them.

### Building Debian packages from Source

**apt-get source [-b] <package>** Download the source Debian package for <package> and extract it. You must have deb-src lines in your /etc/apt/sources.list for this to work. If you supply the `-b' option and you are currently root, then the package will be automatically built if possible.  
  
**apt-get build-dep <package>** Download and install the packages necessary to build the source Debian package <package>. This feature is only present in apt version 0.5 and up. Currently this means that woody and above contain this functionality. If you have an older version of apt then the easiest way to find out the build dependencies is to look in the debian/control file in the source package directory. A common usage of this command is in conjunction with apt-get source -b. For example (as root):

apt-get build-dep <package>   
apt-get source -b <package>

Will download the source package, all of its build dependencies, and attempt to compile the source package.  
  
**dpkg-source -x <package.dsc>** If you have downloaded the source package for a program manually, which includes several files such as a .orig.tar.gz (or .tar.gz if it is Debian native), a .dsc, and a .diff.gz (if it is not Debian native), then you can unpack the source package using this command on the .dsc file.  
  
**dpkg-buildpackage** Builds a Debian package from a Debian source tree. You must be in the main directory of the source tree for this to work. Sample usage:

dpkg-buildpackage -rfakeroot -uc -b

Where `-rfakeroot' instructs it to use the fakeroot program to simulate root privileges (for ownership purposes), `-uc' stands for ``Don't cryptographically sign the changelog'', and `-b' stands for ``Build the binary package only''  
  
**debuild** A handy wrapper script around dpkg-buildpackage that will automatically take care of using fakeroot or not, as well as running lintian and gpg for you.

### Fixing dependencies

**dpkg --configure --pending** If dpkg quits with an error while apt-get install, upgrade, or dist-upgradeing try running this to configure the packages that were already unpacked. Then try apt-get install, upgrade, or dist-upgrade -f, and then try apt-get install, upgrade, or dist-upgrade again. Repeat as needed. This usually resolves most dependency problems (also, if it mentions a specific package for some reason, you might want to try installing or removing that package)  
  
**apt-get install -f   
apt-get upgrade -f   
apt-get dist-upgrade -f** Attempt to fix dependencies while doing one of the above. Note that apt-get install -f does not require a <package> argument.