

[name](#)[synopsis](#)[description](#)[actions](#)[options](#)[original modules](#)[dkms.conf](#)[dkms.conf overrides](#)[/etc/dkms/framework.conf](#)[dkms_autoinstaller](#)[author](#)[webpage](#)

lunar (8) dkms.8.gz

Provided by: [dkms_3.0.9-1_all](#) 🐛

NAME

```
dkms - Dynamic Kernel Module Support
```

SYNOPSIS

```
dkms [action] [options] [module/module-version] [/path/to/source-tree]
      [/path/to/tarball.tar] [/path/to/driver.rpm]
```

DESCRIPTION

dkms is a framework which allows kernel modules to be dynamically built for each kernel on your system in a simplified and organized fashion.

ACTIONS

```
add [module/module-version] [/path/to/source-tree] [/path/to/tarball.tar]
```

Adds a `module/module-version` combination to the tree for builds and installs. If `module/module-version`, `-m module/module-version`, or `-m module -v version` are passed as options, this command requires source in `/usr/src/<module>-<module-version>/` as well as a properly formatted `dkms.conf` file. If `/path/to/source-tree` is passed as an option, and `source-tree` contains a `dkms.conf` file, it will copy `/path/to/source-tree` to `/usr/src/module-module-version`. If `/path/to/tarball.tar` is passed, this command

behaves like the **ldtarball** command.

remove [**module/module-version**] [-k kernel/arch] [--all]

Removes a module/version or module/version/kernel/arch combination from the tree. If the module is currently installed, it first uninstalls it and if applicable, will replace it with its original_module. Use the --all option in order to remove all instances for every kernel at once.

build [**module/module-version**] [-k kernel/arch] [--force]

Builds the specified module/version combo for the specified kernel/arch. If the -k option is not specified it builds for the currently running kernel and arch. All builds occur in the directory /var/lib/dkms/<module>/<module-version>/build/. If the module/module-version combo has not been added, dkms will try to add it, and in that case **build** can take the same arguments that **add** can. If the module is already built, it will not be rebuilt again by default, and the --force option should be used to override this.

unbuild [**module/module-version**] [-k kernel/arch] [--all]

Undoes the build for a module/version or module/version/kernel/arch combination from the tree. If the module is currently installed, it first uninstalls it and if applicable, will replace it with its original_module. Finally all binary kernel modules are removed. Use the --all option in order to remove all instances for every kernel at once.

install [**module/module-version**] [-k kernel/arch] [--force] [/path/to/driver.rpm]

Installs a built module/version combo onto the kernel it was built for. If the kernel option is not specified it assumes the currently running kernel. If the module has not been built, dkms will try to build it. If the module has not been added, dkms will try to add it. In both cases, the **install** command can then take the same arguments as the **build** or **add** commands. If the module is already installed, it will not be reinstalled again by default, and the --force option should be used to override this. If you pass a .rpm file, dkms will try to install that file with **rpm -Uvh** , and it will perform an **autoinstall** action to be sure that everything is built for your kernel if the RPM installed successfully.

uninstall [**module/module-version**] [-k kernel/arch] [--all]

Uninstalls an installed module/module-version combo from the kernel/arch passed in the -k option, or the current kernel if the -k option was not passed. Use the --all option in order to uninstall all instances for every kernel at once. After uninstall completion, the driver will be left in the built state. To completely remove a driver, the remove action should be utilized.

match [--templatekernel kernel/arch] [-k kernel/arch]

Match installs modules onto the specified kernel by looking at the configuration of the specified **templatekernel**. Every module that is installed on the **templatekernel**

within **dkms** is then installed on that specified kernel.

```
mktdarball [module/module-version] [-k kernel/arch] [--archive /path/to/tarball.tar]  
  [--source-only] [--binaries-only]
```

Creates a tarball archive for the specified module/version of all files in the DKMS tree for that module/version combination. This includes the source and any built modules for kernels in the tree (as specified). Otherwise, you can specify a singular kernel to archive only, or multiple kernels to archive (-k kernel1/arch1 -k kernel2/arch2). Optionally, you can use **--archive** to specify the file that you would like to save this tarball to. You can also specify **--binaries-only** if you want the resultant tarball not to include the module source. Likewise, **--source-only** can be used to specify that no prebuilt binaries should be included in the tarball. In general, **mktdarball** is great for systems management purposes as you can build your driver on just one system and then use **ldtarball** on all of your other systems to get the same built modules loaded without having to wait for anything to compile.

```
ldtarball [/path/to/tarball.tar] [--force]
```

This takes a tarball made from the **mktdarball** command and loads it into your DKMS tree. This will leave any newly added modules in the built state and **dkms install** should then be called to install any of them. If files already exist where **ldtarball** is attempting to place them, it will warn and not copy over them. The **--force** option should be used to override this.

```
status [module/module-version] [-k kernel/arch]
```

Returns the current status of modules, versions and kernels within the tree as well as whether they have been added, built or installed. Status can be shown for just a certain module, a certain kernel, a module/version combination or a module/version/kernel combination.

autoinstall

Attempt to install the latest revision of all modules that have been installed for other kernel revisions. `dkms_autoinstaller` is a stub that uses this action to perform its work.

OPTIONS

-m **<module>/<module-version>**

The name of the module and module version you want to operate on. The **-m** part of this option is optional, and can be omitted in virtually all circumstances.

-v **<module-version>**

The version of the module to execute the specified action upon. This option only has to be specified if you pass a **-m** option without a **<module-version>** component of its own.

-k <kernel-version>/<arch>

The kernel and arch to perform the action upon. You can specify multiple kernel version/arch pairs on the command line by repeating the -k argument with a different kernel version and arch. However, not all actions support multiple kernel versions (it will error out in this case). The arch part can be omitted, and DKMS will assume you want it to be the arch of the currently running system.

-a, --arch

The system architecture to perform the action upon. It is optional if you pass it as part of the -k option. If not specified, it assumes the arch of the currently running system (`uname -m`). You can specify multiple arch parameters on the same command line by repeating the -a argument with a different arch name. When multiple architectures are specified, there must be a 1:1 relationship between -k arguments to -a arguments. DKMS will then assume the first -a argument aligns with the first -k kernel and so on for the second, third, etc.

For example, if you were to specify: `-k kernel1 -k kernel2 -a i386 -k kernel3 -a i686 -a x86_64`, DKMS would process this as: `kernel1-i386, kernel2-i686, kernel3-x86_64`.

-q, --quiet

Quiet.

-V, --version

Prints the currently installed version of dkms and exits.

-c <dkms.conf-location>

The location of the `dkms.conf` file. This is needed for the add action and if not specified, it is assumed to be located in `/usr/src/<module>-<module-version>/.` See below for more information on the format of `dkms.conf`.

--config <kernel-.config-location>

During a **build** this option is used to specify an alternate location for the kernel `.config` file which was used to compile that kernel. Normally, **dkms** uses the Red Hat standard location and config filenames located in `/usr/src/linux-<kernel>/configs/.` If the config for the kernel that you are building a module for is not located here or does not have the expected name in this location, you will need to tell **dkms** where the necessary `.config` can be found so that your kernel can be properly prepared for the module build.

--archive <tarball-location>

This option is used during a **ldtarball** action to specify the location of the tarball you wish to load into your DKMS tree. You only have to specify the **--archive** part of this option if `<tarball-location>` does not already exist as a file.

--templatekernel <kernel-version>

This option is required for the action: **match**. Match will look at the templatekernel specified and install all of the same module/version combinations on the other kernel.

--force

This option can be used in conjunction with **ldtarball** to force copying over of extant files.

--binaries-only

This option can be used in conjunction with **mkrtarball** in order to create a DKMS tarball which does not contain the source for the module within it. This can be helpful in reducing the size of the tarball if you know that the system which this tarball will be loaded upon already has the source installed. In order to load a tarball made as binaries-only **you must** have the module source in that systems DKMS tree. If you do not, DKMS **will refuse** to load a binaries-only tarball.

--source-only

This option can be used in conjunction with **mkrtarball** but do not want the tarball you create to have any prebuilt modules within it, passing this option will keep its internal DKMS tarball from containing any prebuilt modules.

--all This option can be used to automatically specify all relevant kernels/arches for a module/module-version. This can be used for things like remove, unbuild and uninstall. This saves the trouble of having to actually specify `-k kernel1 -a arch1 -k kernel2 -a arch2` for every kernel you have built your module for.

--no-depmod

This option prevents DKMS from running the depmod command during **install** and **uninstall** which will avoid (re)calculating module dependencies and thereby save time.

--modprobe-on-install

This option executes modprobe on the modules upon successful installation.

--kernelsourcedir <kernel-source-directory-location>

Using this option you can specify the location of your kernel source directory. Most likely you will not need to set this if your kernel source is accessible via [/lib/modules/\\$kernel version/build](#).

--directive <"cli-directive=cli-value">

Using this option, you can specify additional directives from the command line. The **--directive** option can be used multiple times on the same command-line to specify multiple additional command line directives.

--rpm_safe_upgrade

This flag should be used when packaging DKMS enabled modules in RPMs. It should be specified during both the **add** and **remove** actions in the RPM spec to ensure that DKMS and RPM behave correctly in all scenarios when upgrading between various versions of a dkms enabled module RPM package.

--dkmstree path/to/place

Provides a destination tree for building and installing modules to. Useful in cases that you don't want to contaminate a system when using solely for building.

--sourcetree path/to/place

Provides a location to build a DKMS package from. Useful for systems that you may not have root access, but would still like to be able to build DKMS packages.

--installtree path/to/place

Provides a location to place modules when a `dkms install` command is issued.

-j number

Run no more than `number` jobs in parallel; see the `-j` option of `make(1)`. Defaults to the number of CPUs in the system, detected by `nproc(1)`. Specify `0` to impose no limit on the number of parallel jobs.

ORIGINAL MODULES

During the first install of a module for a `<kernelversion>`, `dkms` will search `/lib/modules/<kernelversion>` for a pre-existing module of the same name. If one is found, it will automatically be saved as an "original_module" so that if the newer module is later removed, `dkms` will put the original module back in its place. Currently, DKMS searches for these original modules with first preference going to modules located in `/lib/modules/<kernelversion>/updates/` followed by `$DEST_MODULE_LOCATION` (as specified in `dkms.conf`). If one cannot be found in either location, a find will be used to locate one for that kernel. If none are found, then during a later uninstall, your kernel will not have that module replaced.

If more than one is found, then the first one located (by preference indicated above) will be considered the "original_module". As well, all copies of the same-named module will be removed from your kernel tree and placed into `/var/lib/dkms/<module>/original_module/$kernelver/collisions` so that they can be *manually* accessible later. DKMS will never actually do anything with the modules found underneath the `/collisions` directory, and they will be stored there until you manually delete them.

DKMS.CONF

When performing an `add`, a proper `dkms.conf` file must be found. A properly formatted conf file is essential for communicating to `dkms` how and where the module should be installed. While not all the directives are required, providing as many as possible helps to limit any ambiguity. Note that the `dkms.conf` is really only a shell-script of variable definitions which are then sourced in by the `dkms` executable (of the format, `DIRECTIVE="directive text goes here"`). As well, the directives are case-sensitive and should be given in **ALL CAPS**.

It is important to understand that many of the DKMS directives are arrays whose index values are tied together. These array associations can be considered families, and there are currently three such families of directive arrays. `MAKE[#]` and `MAKE_MATCH[#]` make up one family. `PATCH[#]` and `PATCH_MATCH[#]` make up the second family. The third and largest family consists of `BUILT_MODULE_NAME[#]`, `BUILT_MODULE_LOCATION[#]`, `DEST_MODULE_NAME[#]`, `DEST_MODULE_LOCATION[#]` and `STRIP[#]`. When indexing these arrays when creating your `dkms.conf`, each family should start at index value `0`.

MAKE[#]=

The MAKE directive array tells DKMS which make command should be used for building your module. The default make command should be put into **MAKE[0]**. Other entries in the MAKE array will only be used if their corresponding entry in **MAKE_MATCH[#]** matches, as a regular expression (using `grep -E`), the kernel that the module is being built for. Note that if no value is placed in **MAKE_MATCH[#]** for any **MAKE[#]** where `# > 0`, then that **MAKE** directive is ignored. **MAKE_MATCH[0]** is optional and if it is populated, it will be used to determine if **MAKE[0]** should be used to build the module for that kernel. If multiple **MAKE_MATCH** directives match against the kernel being built for, the last matching **MAKE[#]** will be used to build your module. If no **MAKE** directive is specified or if no **MAKE_MATCH** matches the kernel being built for, DKMS will attempt to use a generic **MAKE** command to build your module.

`KERNELRELEASE` will be automatically appended to **MAKE[#]**. If you want to suppress this behavior, you can quote the make command: `'make'`.

MAKE_MATCH[#]=

See the above entry on **MAKE[#]** directives. This array should be populated with regular expressions which, when matched against the kernel being built for, will tell **DKMS** to use the corresponding make command in the **MAKE[#]** directive array to build your module.

BUILT_MODULE_NAME[#]=

This directive gives the name of the module just after it is built. If your DKMS module package contains more than one module to install, this is a **required** directive for all of the modules. This directive should explicitly not contain any trailing `".o"` or `".ko"`. Note that for each module within a dkms package, the numeric value of `#` must be the same for each of **BUILT_MODULE_NAME**, **BUILT_MODULE_LOCATION**, **DEST_MODULE_NAME** and **DEST_MODULE_LOCATION** and that the numbering should start at `0` (eg. `BUILT_MODULE_NAME[0]="qla2200"` `BUILT_MODULE_NAME[1]="qla2300"`).

BUILT_MODULE_LOCATION[#]=

This directive tells DKMS where to find your built module after it has been built. This pathname should be given relative to the root directory of your source files (where your `dkms.conf` file can be found). If unset, DKMS expects to find your **BUILT_MODULE_NAME[#]** in the root directory of your source files. Note that for each module within a dkms package, the numeric value of `#` must be the same for each of **BUILT_MODULE_NAME**, **BUILT_MODULE_LOCATION**, **DEST_MODULE_NAME** and **DEST_MODULE_LOCATION** and that the numbering should start at `0` (eg. `BUILT_MODULE_LOCATION[0]="some/dir/"` `BUILT_MODULE_LOCATION[1]="other/dir/"`).

DEST_MODULE_NAME[#]=

This directive can be used to specify the name of the module as it should be installed. This will rename the module from **BUILT_MODULE_NAME[#]** to **DEST_MODULE_NAME[#]**. This directive should explicitly not contain any trailing `".o"` or `".ko"`. If unset, it is assumed to be the same value as **BUILT_MODULE_NAME[#]**. Note that for each module within a dkms package, the numeric value of `#` must be the same for each of **BUILT_MODULE_NAME**, **BUILT_MODULE_LOCATION**,

DEST_MODULE_NAME and DEST_MODULE_LOCATION and that the numbering should start at 0 (eg. DEST_MODULE_NAME[0]="qla2200_6x" DEST_MODULE_NAME[1]="qla2300_6x").

DEST_MODULE_LOCATION[#]=

This directive specifies the destination where a module should be installed to, once compiled. It also is used for finding original_modules. This is a **required** directive, except as noted below. This directive must start with the text "/kernel" which is in reference to `/lib/modules/<kernelversion>/kernel`. Note that for each module within a dkms package, the numeric value of # must be the same for each of BUILT_MODULE_NAME, BUILT_MODULE_LOCATION, DEST_MODULE_NAME and DEST_MODULE_LOCATION and that the numbering should start at 0 (eg. DEST_MODULE_LOCATION[0]="/kernel/drivers/something/" DEST_MODULE_LOCATION[1]="/kernel/drivers/other/").

DEST_MODULE_LOCATION is ignored on Fedora and Red Hat Enterprise Linux, Novell SuSE Linux Enterprise Server 10 and higher, Novell SuSE Linux 10.0 and higher, and Ubuntu. Instead, the proper distribution-specific directory is used.

STRIP[#]=

By default strip is considered to be "yes". If set to "no", DKMS will not run strip -g against your built module to remove debug symbols from it. STRIP[0] is used as the default for any unset entries in the STRIP array.

PACKAGE_NAME=

This directive is used to give the name associated with the entire package of modules. This is the same name that is used with the -m option when building, adding, etc. and may not necessarily be the same as the MODULE_NAME. This directive must be present in every dkms.conf.

PACKAGE_VERSION=

This directive is used to give the version associated with the entire package of modules being installed within that dkms package. This directive must be present in every dkms.conf.

CLEAN= CLEAN specifies the make clean command to be used to clean up both before and after building the module. If unset, it is assumed to be "make clean".

OBSOLETE_BY=

This directive allows you to specify a kernel version that obsoletes the necessity for this particular DKMS module. This can be specified as a particular upstream kernel or an ABI bump of a kernel. For example, "2.6.24" would be an upstream kernel and "2.6.24-16" would represent an ABI bump for a kernel. Both are valid in this area.

Please avoid the use of **OBSOLETE_BY** wherever possible. It's use indicates a lack of proper module versioning using **MODULE_VERSION()** tags in the module source itself. It is better to fix the **MODULE_VERSION()** tags than use **OBSOLETE_BY**. This also introduces a implicit distribution/version dependency on the package, as the value of **OBSOLETE_BY** is meaningful only in the context of a single distribution/version.

If you feel you must use it, please use as such in dkms.conf:

```

ubuntu_804="Ubuntu
8.04"
if [ -x /usr/bin/lsb_release ]; then
    if [ "$( /usr/bin/lsb_release -sir )" == "${ubuntu_804}" ]; then
        OBSOLETE_BY="2.6.25"
    fi
fi

```

PATCH[#]=

Use the **PATCH** directive array to specify patches which should be applied to your source before a build occurs. All patches are expected to be in `-p1` format and are applied with the `patch -p1` command. Each directive should specify the filename of the patch to apply, and all patches must be located in the `patches` subdirectory of your source directory (`/usr/src/<module>-<module-version>/patches/`). If any patch fails to apply, the build will be halted and the rejections can be inspected in `/var/lib/dkms/<module>-<module-version>/build/`. If a **PATCH** should only be applied conditionally, the **PATCH_MATCH[#]** array should be used, and a corresponding regular expression should be placed in **PATCH_MATCH[#]** which will alert dkms to only use that **PATCH[#]** if the regular expression matches the kernel which the module is currently being built for.

PATCH_MATCH[#]=

See the above description for **PATCH[#]** directives. If you only want a patch applied in certain scenarios, the **PATCH_MATCH** array should be utilized by giving a regular expression which matches the kernels you intend the corresponding **PATCH[#]** to be applied to before building that module.

AUTOINSTALL=

If this directive is set to **yes** then the service `/etc/rc.d/init.d/dkms_autoinstaller` will automatically try to install this module on any kernel you boot into. See the section on **dkms_autoinstaller** for more information.

BUILD_DEPENDS[#]=

This optional directive is an array that allows you to specify other modules as dependencies for your module. Each array element should be the **PACKAGE_NAME** of another module that is managed by dkms. Do not specify a version or architecture in the dependency. Note that this directive is only advisory; missing or broken dependencies cause non-fatal warnings.

BUILD_EXCLUSIVE_KERNEL=

This optional directive allows you to specify a regular expression which defines the subset of kernels which DKMS is allowed to build your module for. If the kernel being built for does not match against this regular expression, the dkms build will error out. For example, if you set it as `"^2.4.*"`, your module would not be built for 2.6 kernels.

BUILD_EXCLUSIVE_ARCH=

This optional directive functions very similarly to **BUILD_EXCLUSIVE_KERNEL** except that it matches against the kernel architecture. For example, if you set it to

= "i.86", your module would not be built for ia32e, x86_64, amd64, s390, etc.

POST_ADD=

The name of the script to be run after an **add** is performed. The path should be given relative to the root directory of your source.

POST_BUILD=

The name of the script to be run after a **build** is performed. The path should be given relative to the root directory of your source.

POST_INSTALL=

The name of the script to be run after an **install** is performed. The path should be given relative to the root directory of your source.

POST_REMOVE=

The name of the script to be run after a **remove** is performed. The path should be given relative to the root directory of your source.

PRE_BUILD=

The name of the script to be run before a **build** is performed. The path should be given relative to the root directory of your source.

PRE_INSTALL=

The name of the script to be run before an **install** is performed. The path should be given relative to the root directory of your source. If the script exits with a non-zero value, the install will be aborted. This is typically used to perform a custom version comparison.

DKMS.CONF VARIABLES

Within your dkms.conf file, you can use certain variables which will be replaced at run-time with their values.

\$kernelver

This variable can be used within a directive definition and during use, the actual kernel version in question will be substituted in its place. This is especially useful in MAKE commands when specifying which INCLUDE statements should be used when compiling your module (eg. MAKE="make all INCLUDEDIR=[/lib/modules/\\${kernelver}/build/include](#)").

\$kernel_source_dir

This variable holds the value of the location of your kernel source directory. Usually, this will be [/lib/modules/\\$kernelver/build](#), unless otherwise specified with the **--kernelourcedir** option.

DKMS.CONF OVERRIDES

You can override the module-provided dkms.conf files. Every time after a dkms.conf file is read, dkms will look for and read the following files in order:

/etc/dkms/<module>.conf

/etc/dkms/<module>-<module-version>.conf

/etc/dkms/<module>-<module-version>-<kernel>.conf

/etc/dkms/<module>-<module-version>-<kernel>-<arch>.conf

You can use these files to override settings in the module-provided dkms.conf files.

/etc/dkms/framework.conf

This configuration file controls how the overall DKMS framework handles. It is sourced in every time the dkms command is run. Mainly it can currently be used to set different default values for the variables.

The file contains descriptions for each directive it supports.

Additionally to /etc/dkms/framework.conf, **any file matching the glob** /etc/dkms/framework.conf.d/*.conf will be loaded as well.

\$dkms_tree, \$source_tree, \$install_tree, \$tmp_location

Control which folders DKMS uses for components and artifacts.

\$verbose

Can be set to anything but a null value to enable verbose output in DKMS.

\$symlink_modules

Controls whether binary modules are copied to [/lib/modules](#) or if only symlinks are created there. Note that these variables can also be manipulated on the command line with --dkmtree, --sourcetree, --installtree and --symlink-modules options.

\$autoinstall_all_kernels

Used by the common postinst for DKMS modules. It controls if the build should be done for all installed kernels or only for the current and latest installed kernel. It has no command line equivalent.

\$sign_file

This is the path of the [sign-file](#) kernel binary that is used to sign the kernel modules. The variable **\$kernelver** can be used in path to represent the target kernel version. The path for the binary depends on the distribution.

\$mok_signing_key, \$mok_certificate

Location of the key and certificate files used for Secure boot. The variable **\$kernelver** can be used in path to represent the target kernel version. [mok_signing_key](#) can also be a "pkcs11:..." string for PKCS#11 engine, as long as the sign_file program supports it.

\$modprobe_on_install

Automatically load the built modules upon successful installation.

dkms_autoinstaller

This boot-time service automatically installs any module which has **AUTOINSTALL="yes"** set in its **dkms.conf** file. The service works quite simply and if multiple versions of a module are in your system's DKMS tree, it will not do anything and instead explain that manual intervention is required.

AUTHOR

Gary Lerhaupt, Emil Velikov, Simone Caronni, Xu Zhen

WEBPAGE

<https://github.com/dell/dkms>

Powered by the [Ubuntu Manpage Repository](#), file bugs in [Launchpad](#)

© 2019 Canonical Ltd. Ubuntu and Canonical are registered trademarks of Canonical Ltd.