Simplifying Linux Management with

Dynamic Kernel Module Support

At times, administrators may need newer drivers than the ones found in the Linux® operating system kernel. Dynamic Kernel Module Support, a software project created by the Dell™ Linux Engineering team, efficiently decouples driver releases from kernel releases, helping to provide an orderly method for distributing the latest drivers even when they are not yet merged into the Linux kernel.

BY GARY LERHAUPT AND MATT DOMSCH

As the Linux® operating system (OS) gains a deeper foothold in enterprise environments, system administrators have become increasingly concerned about the management of Linux kernel modules. In the best-case scenario, every driver needed to run every piece of system hardware would come precompiled with the Linux kernel. In practice, however, hardware drivers often are released separately from the kernel, and updates for drivers native to the kernel also are released independently, superseding the drivers within the latest kernel version.

Until the ultimate goal of pushing all driver modifications back into the kernel is met, Dynamic Kernel Module Support (DKMS), a software project created by the Dell™ Linux Engineering team, can help administrators add, build, install, remove, and track Linux kernel modules. DKMS aims to create a standardized framework for collecting driver source code, building this source code into loadable compiled-module binary files, and then installing and uninstalling these modules into the Linux kernel as needed. In addition, DKMS provides powerful features for managing and maintaining modules across multiple systems and keeping track of which module version is installed on which kernel version.

By creating a separate framework for driver source code and the module binary files that are compiled from that source code, DKMS efficiently decouples driver releases from kernel releases. Decoupling driver and kernel releases permits administrators to update drivers on existing kernels in an orderly and supportable manner as soon as they are available. Thus, DKMS serves as a stopgap, providing a way to distribute the latest driver updates until the source code can be merged back into the kernel.

In addition, DKMS streamlines the process of compiling from source code. Rebuilding RPM™ (Red Hat® Package Manager) source packages can be time-consuming and problematic. DKMS helps simplify Linux development by creating a single executable that can be called to build, install, or uninstall modules.

In the best-case scenario, every driver needed to run every piece of system hardware would come precompiled with the Linux kernel.

Further, DKMS makes configuring modules on new kernels particularly

easy for less-experienced Linux developers: The modules to be installed can be based solely on the configuration of a kernel that was previously running. In production environments, this represents an immediate advantage. For example, using DKMS, IT managers no longer have to choose between a predefined solution stack or the security enhancements of a newer kernel.

Decoupling driver and kernel releases permits administrators to update drivers on existing kernels in an orderly and supportable manner as soon as they are available.

DKMS has two target audiences: developers who maintain and package drivers, and system administrators. This article focuses on DKMS from the system administrator perspective of using DKMS to simplify Linux enterprise computing management.[1]

Understanding basic DKMS commands

Before exploring the uses of DKMS, it is helpful to understand the life cycle by which DKMS maintains kernel modules. Figure 1 represents each potential state for a module—Added, Built, and Installed—and each arrow indicates a DKMS action that can be used to switch between the various states. The sections that follow examine each of these DKMS actions further.

Most importantly, DKMS was designed to work with RPM. Using DKMS to install a kernel module often can be as easy as installing a DKMS-enabled module RPM, because module packagers can use DKMS to add, build, and install modules within RPM packages. Wrapping DKMS commands inside an RPM package preserves the benefits of RPM—package versioning, security, dependency resolution, and package distribution methodologies—while DKMS handles the work that RPM does not: the versioning and building of individual kernel modules. Of course, DKMS works just as well when not used in conjunction with RPM, so it is important to understand how to use these basic commands to fully leverage the capabilities of DKMS.

Add command adds a module and module version to the tree

DKMS manages kernel modules at the source-code level. First, the module source code must be located in the directory /usr/src/*module-module-version* on the build system. A dkms.conf file with appropriately formatted directives also must reside within this configuration file to tell DKMS where to install the module and how to build it. The dkms.conf file should come from the module packager and be included with the module source code. Once these two requirements have been met and DKMS has been installed on a system, administrators can begin using DKMS by adding a module and module version to the DKMS tree. For example:

```
dkms add -m megaraid2 -v 2.00.9
```

This sample `add` command would add megaraid2/2.00.9 to the already existing /var/dkms tree, leaving the module in an Added state.

Build command compiles the module

Once in the Added state, the module is ready to be built using the DKMS `build` command. The `build` command requires that the proper kernel source code be located on the system in the /lib/module/*kernel-version*/build directory. The `make` command that is used to compile the module is specified in the dkms.conf configuration file. The following sample `build` command continues the megaraid2/2.00.9 example:

```
dkms build -m megaraid2 -v 2.00.9 -k 2.4.21-4.ELsmp
```

The `build` command compiles the module but stops short of installing it. As this example indicates, the `build` command expects a kernel-version parameter. If this kernel name is left out, it assumes the currently running kernel. The `build` command also can build modules for kernels that are not currently running; this functionality is provided through use of a kernel preparation subroutine that runs before any module build is performed. The subroutine ensures that the module being built is linked against the proper kernel symbols.

In this example, successful completion of a build creates the /var/dkms/megaraid2/2.00.9/2.4.21-4.ELsmp directory as well as the log and module subdirectories within this directory. The log directory holds a log file of the module make and the module directory holds copies of the resultant .o binary files that were compiled.

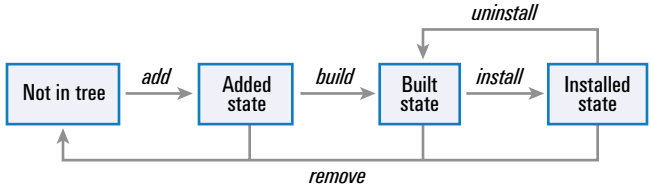

Figure 1. The DKMS life cycle

[1] For a detailed discussion about creating and developing DKMS-enabled module packages from the developer's perspective, see "Exploring Dynamic Kernel Module Support" in *Linux Journal*, September 2003, http://www.linuxjournal.com/article.php?sid=6896.

Install command copies the compiled module binary files to the kernel tree

Upon completion of a build, the module can be installed on the kernel for which it was built. The `install` command copies the compiled module binary files to the correct location in the /lib/modules tree as specified in the dkms.conf file. If a module by that name already resides in that location, DKMS saves the existing module in the /var/dkms/*module-name*/original_module directory. This process helps ensure that the older module can be put back into place if, at a later date, the newer module is uninstalled. A sample `install` command is as follows:

```
dkms install -m megaraid2 -v 2.00.9 -k 2.4.21-4.ELsmp
```

In this example, if an original megaraid2 module existed within the 2.4.21-4.ELsmp kernel, it would be saved to /var/dkms/megaraid2/original_module/2.4.21-4.ELsmp.

Uninstall and remove commands expunge modules to differing degrees

The DKMS life cycle also enables administrators to uninstall or remove a module from the tree. The `uninstall` command removes the installed module and, if applicable, replaces it with the original module. When multiple versions of a module are located within the DKMS tree, if one version is uninstalled, DKMS does not try to determine which of these other versions to put in its place. Instead, if a true "original_module" was saved from the very first DKMS installation, it will be put back into the kernel and all other versions of that module will be left in the Built state. A sample `uninstall` command is as follows:

```
dkms uninstall -m megaraid2 -v 2.00.9
    -k 2.4.21-4.ELsmp
```

Again, if the kernel-version parameter is unset, the currently running kernel is assumed. However, this same behavior does not occur with the `remove` command. Although the `remove` and `uninstall` commands are similar, some important differences exist. The `remove` command uninstalls, but also is used to clean the DKMS tree. If the module version being removed is the last instance of that module version for all kernels on a system, after the uninstall portion of the `remove` command completes, the `remove` command will physically delete all traces of that module from the DKMS tree. That is, when the `uninstall` command completes, modules are left in the Built state; when the `remove` command completes, an administrator would have to start over from the `add` command before being able to again use the module with DKMS. Two sample `remove` commands are shown here:

```
dkms remove -m megaraid2 -v 2.00.9
    -k 2.4.21-4.ELsmp
dkms remove -m megaraid2 -v 2.00.9 --all
```

DKMS serves as a stopgap, providing a way to distribute the latest driver updates until the source code can be merged back into the kernel.

The first sample command would uninstall the module; if this module and module version were not installed on any other kernel, the command would remove the module from the DKMS tree altogether. If, however, megaraid2/2.00.9 module and module version also were installed on the 2.4.21-4.ELhugemem kernel, the first `remove` command would leave the module alone, and thus it would remain intact in the DKMS tree. Because the second sample command contains the `--all` parameter, not the `-k` *`kernel`* parameter, the second command would uninstall all versions of the megaraid2/2.00.9 module from all kernels and then completely expunge any references of megaraid2/2.00.9 from the DKMS tree.

Extending DKMS functionality with auxiliary commands and services

The `add`, `build`, `install`, `uninstall`, and `remove` commands—which correlate to the DKMS life cycle—are the fundamental DKMS commands. The auxiliary DKMS functionality discussed in this section extends and improves upon the capabilities of these basic commands.

Status command returns data about modules currently located in the tree

DKMS also includes a fully functional `status` command that returns information about the modules and module versions currently located in the tree. The specificity of the information returned depends on which parameters are passed to the `status` command. If no parameters are set, this command will return all information found. Each status entry will return output—`added`, `built`, or `installed`—to indicate the state; and if an original module has been saved, this information also will be displayed. Several sample `status` commands are shown here:

```
dkms status
dkms status -m megaraid2
dkms status -m megaraid2 -v 2.00.9
dkms status -k 2.4.21-4.ELsmp
dkms status -m megaraid2 -v 2.00.9
    -k 2.4.21-4.ELsmp
```

Match command applies module configurations from one kernel to another

Another major feature of DKMS is the `match` command. The `match` command takes the configuration of a DKMS-installed module for one kernel and applies the same configuration to another kernel.

When the `match` command completes, the same module and module versions that were installed for one kernel are installed on the other kernel. This is helpful to administrators who are upgrading from an existing kernel to a newer kernel, but would like to keep the same DKMS modules in place for the new kernel. A sample `match` command is as follows:

```
dkms match --templatekernel 2.4.21-4.ELsmp
   -k 2.4.21-5.ELsmp
```

As shown in the preceding example, the `--templatekernel` parameter is the kernel on which the configuration is based, while `-k` is the kernel upon which the configuration is instated.

Dkms_autoinstaller service automatically installs a designated module

The dkms_autoinstaller service is similar in behavior to the `match` command. This service is installed in the /etc/init.d directory as part of the DKMS RPM. If an autoinstall parameter is set within the dkms.conf configuration file in a module, that module is eligible for the dkms_autoinstaller service to automatically, upon booting, build it into a new kernel. When the administrator later boots a system into a new kernel, the dkms_autoinstaller service will then automatically build and install modules designated for use with this service.

Mkdriverdisk command creates a driver disk image

The final auxiliary DKMS command is `mkdriverdisk`. As its name suggests, the `mkdriverdisk` command builds modules to create a driver disk image for use in distributing updated drivers to Linux installations. A sample `mkdriverdisk` command might look like this:

```
dkms mkdriverdisk -d redhat -m megaraid2
   -v 2.00.9 -k 2.4.21-4.ELBOOT
```

Currently, the only supported distribution driver disk format is Red Hat. For more information on the extra necessary files and their required formats for DKMS to create Red Hat driver disks or for general information on Red Hat driver disks, see http://people.redhat.com/dledford. When creating driver disks with DKMS, administrators should place these files in a subdirectory underneath the module source directory: for example, /usr/src/*module-module-version*/redhat_driver_disk.

Managing multiple systems using mktarball and ldtarball commands

As the preceding examples demonstrate, DKMS provides a simple mechanism to build, install, and track driver updates. This functionality not only is applicable to stand-alone machines, but also is useful for IT departments that administer multiple similar servers. The DKMS `mktarball` and `ldtarball` commands enable organizations having a compiler and kernel source on only one system—a master build system—to deploy a new driver to multiple additional systems.

DKMS helps simplify Linux development by creating a single executable that can be called to build, install, or uninstall modules.

The `mktarball` command packages copies of each .o binary file from the module directory that was compiled using the DKMS `build` command into a compressed tar file. This compressed tar file may then be copied to each target system. Administrators can use the DKMS `ldtarball` command to load the compressed tar files into a DKMS tree, leaving each module in the Built state, ready to be installed. The `mktarball` and `ldtarball` commands keep administrators from having to install both kernel source code and compilers on every target system.

The following example assumes that an administrator has built the megaraid2 driver, version 2.00.9, for two different kernel families—2.4.20-9 and 2.4.21-4.EL—on a master build system:

```
# dkms status
megaraid2, 2.00.9, 2.4.20-9: built
megaraid2, 2.00.9, 2.4.20-9bigmem: built
megaraid2, 2.00.9, 2.4.20-9BOOT: built
megaraid2, 2.00.9, 2.4.20-9smp: built
megaraid2, 2.00.9, 2.4.21-4.EL: built
megaraid2, 2.00.9, 2.4.21-4.ELBOOT: built
megaraid2, 2.00.9, 2.4.21-4.ELhugemem: built
megaraid2, 2.00.9, 2.4.21-4.ELsmp: built
```

To deploy this version of the driver to several systems without rebuilding from the source code each time, an administrator can use the `mktarball` command to generate two compressed tar files—one for each kernel family:

```
# dkms mktarball -m megaraid2 -v 2.00.9
   -k 2.4.21-4.EL,2.4.21-4.ELsmp,2.4.21-4.ELBOOT,
   2.4.21-4.ELhugemem

Marking /usr/src/megaraid2-2.00.9 for archiving...
Marking kernel 2.4.21-4.EL for archiving...
Marking kernel 2.4.21-4.ELBOOT for archiving...
Marking kernel 2.4.21-4.ELhugemem for archiving...
Marking kernel 2.4.21-4.ELsmp for archiving...
Tarball location: /var/dkms/megaraid2/2.00.9/
   tarball/megaraid2-2.00.9-kernel2.4.21-4.EL-
   kernel2.4.21-4.ELBOOT-kernel2.4.21-4.ELhugemem-
   kernel2.4.21-4.ELsmp.dkms.tar.gz
Done.
```

When one large compressed tar file that contains modules for both families is preferred, administrators can omit the `-k` parameter and kernel list; DKMS then will include a module for every kernel version found.

After creating one or more compressed tar files, administrators should run the `status` command to ensure that the target DKMS tree does not already contain the modules to be loaded:

```
# dkms status
Nothing found within the DKMS tree for this
    status command.
If your modules were not installed with DKMS,
    they will not show up here.
```

Next, the compressed tar file can be renamed, if desired, and copied to each of the target systems using any mechanism. The compressed tar file is then loaded on the target system:

```
# dkms ldtarball --archive=megaraid2-2.00.9-
    kernel2.4.21-4.EL-kernel2.4.21-4.ELBOOT-
    kernel2.4.21-4.ELhugemem-kernel2.4.21-4.ELsmp.dk
    ms.tar.gz
Loading tarball for module: megaraid2 / version:
    2.00.9
Loading /usr/src/megaraid2-2.00.9...
Loading /var/dkms/megaraid2/2.00.9/2.4.21-4.EL...
Loading /var/dkms/megaraid2/2.00.9/2.4.21-
    4.ELBOOT...
Loading /var/dkms/megaraid2/2.00.9/2.4.21
    -4.ELhugemem...
Loading /var/dkms/megaraid2/2.00.9/2.4.21-4.ELsmp...
Creating /var/dkms/megaraid2/2.00.9/source symlink...
```

The DKMS `ldtarball` command leaves modules in the Built state, not the Installed state. Administrators should verify both that the modules are present and that they are in the Built state:

```
# dkms status
megaraid2, 2.00.9, 2.4.21-4.EL: built
megaraid2, 2.00.9, 2.4.21-4.ELBOOT: built
megaraid2, 2.00.9, 2.4.21-4.ELhugemem: built
megaraid2, 2.00.9, 2.4.21-4.ELsmp: built
```

The preceding steps must be repeated for each kernel version into which modules are to be installed.

Simplifying administration and increasing system stability with DKMS

DKMS can simplify Linux system administration by providing a versioning framework for installing driver modules on kernels. DKMS integrates with RPM for package distribution and installation, facilitates OS installation on new hardware, and helps maintain driver consistency across multiple servers. By enabling deployment of driver updates independent of kernel updates, DKMS reduces the scope of change for configuration management, and thus can help increase system stability.

DKMS is licensed under the GNU General Public License (GPL). Interested parties may contribute to its development by signing up for the dkms-devel@lists.us.dell.com mailing list located at http://lists.us.dell.com. DKMS can be downloaded from http://linux.dell.com/dkms.

Gary Lerhaupt (gary_lerhaupt@dell.com) is a software engineer on the Linux Engineering Team of the Dell Product Group, and is the author of the Dynamic Kernel Module Support project. Gary is a Red Hat Certified Engineer (RHCE) and has a B.S. in Computer Science and Engineering from The Ohio State University.

Matt Domsch (matt_domsch@dell.com) is a lead and senior engineer on the Linux Engineering Team of the Dell Product Group, which tests Linux on all Dell PowerEdge™ servers. Matt has an M.S. in Computer Science from Vanderbilt University and a B.S. in Computer Science and Engineering from the Massachusetts Institute of Technology. His primary areas of interest include networking and operating systems.

FOR MORE INFORMATION

DKMS project home page:
http://linux.dell.com/dkms

DKMS mailing list:
http://lists.us.dell.com

DKMS information for driver maintainers:
http://www.linuxjournal.com/article.php?sid=6896

Red Hat driver disk reference:
http://people.redhat.com/dledford