Loading/Unloading Kernel Modules

coursera.org/learn/linux-for-developers/supplement/zBpFQ/loading-unloading-kernel-modules

Many facilities in the Linux kernel can either be built-in to the kernel when it is initially loaded, or dynamically added (or removed) later as modules, upon need or demand. Indeed, all but some of the most central kernel components are designed to be modular.

Such modules may or may not be device drivers; for example, they may implement a certain network protocol or filesystem. Even in cases where the functionality will virtually always be needed, incorporation of the ability to load and unload as a module facilitates development, as kernel reboots are not required to test changes.

Even with the widespread usage of kernel modules, Linux retains a monolithic kernel architecture, rather than a microkernel one. This is because once a module is loaded, it becomes a fully functional part of the kernel, with few restrictions. It communicates with all kernel sub-systems primarily through shared resources, such as memory and locks, rather than through message passing as might a microkernel.

Linux is hardly the only operating system to use modules; certainly Solaris does it, as well as does AIX, which terms them kernel extensions. However, Linux uses them in a particularly robust fashion.

Module loading and unloading must be done as the root user. If you know the full path name, you can always load the module directly with:

```
$ sudo /sbin/insmod <pathto>/module_name.ko
```

A kernel module always has a file extension of **.ko**, as in **e1000e.ko**, **ext4.ko**, or **nouveau.ko**.

Many modules can be loaded while specifying parameter values, such as in:

```
 $ sudo /sbin/insmod <pathto>/module_name.ko irq=12 debug=3
```

While the module is loaded, you can always see its status with the **lsmod** command:

```
$ lsmod

Module              Size      Used by

coretemp            16384      0

e1000e            237568      0

ptp               20480      1  e1000e

pps_core            20480      1  ptp
```

Direct removal can always be done with:

```
$ sudo /sbin/rmmod module_name
```

Note that it is not necessary to supply the full path name or the **.ko** extension when removing a module.

In most circumstances, the **insmod** and **rmmod** commands are not usually used to load/unload modules, but rather the **modprobe** command is, as in:

```
$ sudo /sbin/modprobe module_name

$ sudo /sbin/modprobe -r module_name
```

with the second form being used for removal. For **modprobe** to work, the modules must be installed in the proper location, generally under **/lib/modules/$(uname -r)** where **$(uname -r)** gives the current kernel version, such as 4.18.3.

You can also pass parameters to **modprobe** in the exact same fashion as you do with **insmod**, as in:

```
$ sudo /sbin/modprobe module_name.ko irq=12 debug=3
```

There are some important things to keep in mind when loading and unloading modules:

- It is impossible to unload a module that is being used by another module, which can be seen from the **lsmod** listing.
- It is impossible to unload a module that is being used by one or more processes, which can also be seen from the **lsmod** listing. However, there are modules which do not always keep track of this reference count, such as network device driver modules, as it would make it too difficult to temporarily replace a module without shutting down and restarting much of the whole network stack.

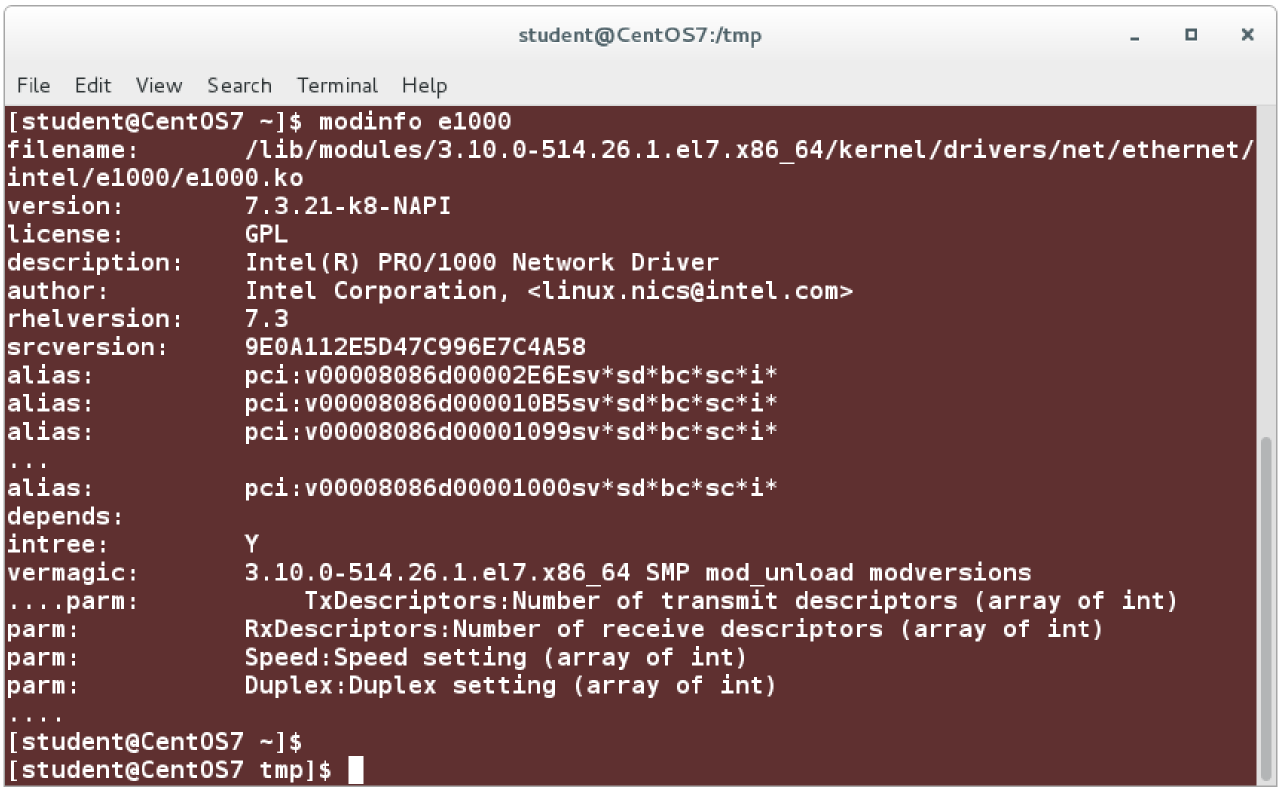
- When a module is loaded with **modprobe**, the system will automatically load any other modules that are required to be loaded first.
- When a module is unloaded with **modprobe -r**, the system will automatically unload any other modules being used by the module, if they are not being simultaneously used by any other loaded modules.
- Files in the **/etc/modprobe.d** directory control some parameters that come into play when loading with **modprobe**. These parameters include module name aliases and automatically supplied options. This directory also contains information about blacklisted modules, which should never be located and loaded.

The **modinfo** command can be used to find out information about kernel modules, whether they are loaded or not, as in:

```
$ /sbin/modinfo my_module

$ /sbin/modinfo <pathto>/my_module.ko
```

An example can be seen in the screenshot below.



```
[student@CentOS7 ~]$ modinfo e1000
filename:       /lib/modules/3.10.0-514.26.1.el7.x86_64/kernel/drivers/net/ethernet/
intel/e1000/e1000.ko
version:        7.3.21-k8-NAPI
license:        GPL
description:    Intel(R) PRO/1000 Network Driver
author:         Intel Corporation, <linux.nics@intel.com>
rhelversion:    7.3
srcversion:     9E0A112E5D47C996E7C4A58
alias:          pci:v00008086d00002E6Esv*sd*bc*sc*i*
alias:          pci:v00008086d000010B5sv*sd*bc*sc*i*
alias:          pci:v00008086d00001099sv*sd*bc*sc*i*
...
alias:          pci:v00008086d00001000sv*sd*bc*sc*i*
depends:
intree:         Y
vermagic:       3.10.0-514.26.1.el7.x86_64 SMP mod_unload modversions
....parm:           TxDescriptors:Number of transmit descriptors (array of int)
parm:           RxDescriptors:Number of receive descriptors (array of int)
parm:           Speed:Speed setting (array of int)
parm:           Duplex:Duplex setting (array of int)
....
[student@CentOS7 ~]$
[student@CentOS7 tmp]$
```