Linux Kernel Modules - Load, Unload, Configure

Linux allows the Kernel to be configured at run time, to enable or disable different services as you see fit. This way you don't have to compile a monolithic kernel, and can save some memory usage. Some modules you'll only need for a short time, others you'll need all the time. You can configure your Linux machine to load kernel modules on startup so you don't have to remember to do that when (if) you reboot.

The Linux Hardware Devices has more information about modules and kernel configuration options.

Module Commands

There are a few commands that allow you to maniuplate the kernel. Each is quickly described below, for more information say `man [command]`.

- **depmod** - handle dependency descriptions for loadable kernel modules.
- **insmod** - install loadable kernel module.
- **lsmod** - list loaded modules.
- **modinfo** - display information about a kernel module.
- **modprobe** - high level handling of loadable modules.
- **rmmod** - unload loadable modules.

The usage of the commands is demonstrated below, it is left as an excerise to the reader to fully understand the commands.

Using Module Commands

Below the different kernel module commands are demonstrated

```
# Show the module dependencies.
depmod -n

# Install some module
insmod --autoclean [modnam]

# This lists all currently loaded modules, lsmod takes no useful parameters
lsmod

# Display information about module eepro100
modinfo --author --description --parameters eepro100

# Removing a module (don't use the example)
```

```
rmmod --all --stacks ip_tables
```

Module Configuration Files

The kernel modules can use two different methods of automatic loading. The first method (modules.conf) is my preferred method, but you can do as you please.

- **modules.conf** - This method load the modules before the rest of the services, I think before your computer chooses which runlevel to use
- **rc.local** - Using this method loads the modules after all other services are started

Using 'modules.conf' will require you to say `man 5 modules.conf`. Using 'rc.local' requires you to place the necessary commands (see above) in the right order.

Sample modules.conf

```
# modules.conf - configuration file for loading kernel modules
# Create a module alias parport_lowlevel to parport_pc
alias parport_lowlevel parport_pc
# Alias eth0 to my eepro100 (Intel Pro 100)
alias eth0 eepro100
# Execute /sbin/modprobe ip_conntrack_ftp after loading ip_tables
post-install ip_tables /sbin/modprobe ip_conntrack_ftp
# Execute /sbin/modprobe ip_nat_ftp after loading ip_tables
post-install ip_tables /sbin/modprobe ip_nat_ftp
```

Sample rc.local

```
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.

/sbin/insmod ip_tables
/sbin/modprobe ip_conntrack_ftp
/sbin/modprobe ip_nat_ftp
```

Conclusion

You should see/know that modules are necessary. They can be loaded via 'modules.conf' or 'rc.local', but 'modules.conf' load them first and 'rc.local' loads them

last. Using the various module commands you can add, remove, list or get information about modules.