NAME

modprobe.d, modprobe.conf − Configuration directory/file for modprobe

DESCRIPTION

Because the **modprobe** command can add or remove more than one module, due to module dependencies, we need a method of specifying what options are to be used with those modules. All files underneath the */etc/modprobe.d* directory which end with the *.conf* extension specify those options as required. (the */etc/modprobe.conf* file can also be used if it exists, but that will be removed in a future version). They can also be used to create convenient aliases: alternate names for a module, or they can override the normal **modprobe** behavior altogether for those with special requirements (such as inserting more than one module).

Note that module and alias names (like other module names) can have - or _ in them: both are interchangable throughout all the module commands.

The format of and files under *modprobe.d* and */etc/modprobe.conf* is simple: one command per line, with blank lines and lines starting with '#' ignored (useful for adding comments). A '\' at the end of a line causes it to continue on the next line, which makes the file a bit neater.

COMMANDS

alias *wildcard modulename*

This allows you to give alternate names for a module. For example: "alias my-mod really_long_modulename" means you can use "modprobe my-mod" instead of "modprobe really_long_modulename". You can also use shell-style wildcards, so "alias my-mod* really_long_modulename" means that "modprobe my-mod-something" has the same effect. You can't have aliases to other aliases (that way lies madness), but aliases can have options, which will be added to any other options.

Note that modules can also contain their own aliases, which you can see using **modinfo**. These aliases are used as a last resort (ie. if there is no real module, **install**, **remove**, or **alias** command in the configuration).

options *modulename option...*

This command allows you to add options to the module *modulename* (which might be an alias) every time it is inserted into the kernel: whether directly (using **modprobe** *modulename* or because the module being inserted depends on this module.

All options are added together: they can come from an **option** for the module itself, for an alias, and on the command line.

install *modulename command...*

This is the most powerful primitive: it tells **modprobe** to run your command instead of inserting the module in the kernel as normal. The command can be any shell command: this allows you to do any kind of complex processing you might wish. For example, if the module "fred" works better with the module "barney" already installed (but it doesn't depend on it, so **modprobe** won't automatically load it), you could say "install fred /sbin/modprobe barney; /sbin/modprobe --ignore-install fred", which would do what you wanted. Note the **--ignore-install**, which stops the second **modprobe** from running the same **install** command again. See also **remove** below.

You can also use **install** to make up modules which don't otherwise exist. For example: "install probe-ethernet /sbin/modprobe e100 || /sbin/modprobe eepro100", which will first try to load the e100 driver, and if it fails, then the eepro100 driver when you do "modprobe probe-ethernet".

If you use the string "$CMDLINE_OPTS" in the command, it will be replaced by any options specified on the modprobe command line. This can be useful because users expect "modprobe fred opt=1" to pass the "opt=1" arg to the module, even if there's an install command in the configuration file. So our above example becomes "install fred /sbin/modprobe barney; /sbin/modprobe --ignore-install fred $CMDLINE_OPTS"

remove *modulename command...*

This is similar to the **install** command above, except it is invoked when "modprobe -r" is run. The removal counterparts to the two examples above would be: "remove fred /sbin/modprobe -r --ignore-remove fred && /sbin/modprobe -r barney", and "remove probe-ethernet /sbin/modprobe -r eepro100 || /sbin/modprobe -r e100".

blacklist *modulename*

Modules can contain their own aliases: usually these are aliases describing the devices they support, such as "pci:123...". These "internal" aliases can be overridden by normal "alias" keywords, but there are cases where two or more modules both support the same devices, or a module invalidly claims to support a device: the **blacklist** keyword indicates that all of that particular module's internal aliases are to be ignored.

COPYRIGHT

SEE ALSO

modprobe(8), **modules.dep**(5)