NAME

/etc/network/interfaces − network interface configuration for ifup and ifdown

DESCRIPTION

/etc/network/interfaces contains network interface configuration information for the **ifup**(8) and **ifdown**(8) commands. This is where you configure how your system is connected to the network.

EXAMPLE

The following example configures two network interfaces: eth0 is brought up at boot, and uses DHCP for IPv4 and SLAAC for IPv6, whereas eth1 is brought up whenever the network hardware is detected, and is configured with static IPv4 and IPv6 addresses.

```
auto eth0
allow-hotplug eth1

iface eth0 inet dhcp

iface eth0 inet6 auto

iface eth1 inet static
      address 192.168.1.2/24
      gateway 192.168.1.1

iface eth1 inet6 static
      address fec0:0:0:1::2/64
      gateway fec0:0:0:1::1
```

FILE FORMAT

Lines starting with '#' are ignored. Note that end-of-line comments are NOT supported, comments must be on a line of their own.

A line may be extended across multiple lines by making the last character a backslash.

The file consists of zero or more "iface", "mapping", "auto", "allow-", "rename", "source" and "source-directory" stanzas. These will be described in more detail in the following sections.

INTERFACE SELECTION

Lines beginning with the word "auto" are used to identify the physical interfaces to be brought up when **ifup** is run with the **−a** option. (This option is also used by the system boot scripts, so interfaces marked "auto" are brought up at boot time.) Physical interface names should follow the word "auto" on the same line. There can be multiple "auto" stanzas. **ifup** brings the named interfaces up in the order listed.

Lines beginning with "allow-" are used to identify interfaces that should be brought up automatically by various subsystems. This may be done using a command such as "ifup −−allow=hotplug eth0 eth1", which will only bring up eth0 or eth1 if it is listed in an "allow-hotplug" line. Note that "allow-auto" and "auto" are synonyms. (Interfaces marked "allow-hotplug" are brought up when udev detects them. This can either be during boot if the interface is already present, or at a later time, for example when plugging in a USB network card. Please note that this does not have anything to do with detecting a network cable being plugged in.)

Lines beginning with "no-auto-down" are used to identify interfaces that should not be brought down by the command "ifdown -a". Its main use is to prevent an interface from being brought down during system shutdown time, for example if the root filesystem is a network filesystem and the interface should stay up until the very end. Note that you can still bring down the interface by specifying the interface name explicitly.

Lines beginning with "no-scripts" are used to identify interfaces for which scripts in */etc/network/if−*.d/* should not be run when those interfaces are brought up or down. he above will match eth0 and eth1, and will bring up both interfaces using the "iface eth" stanza.

INTERFACE RENAMING

Lines beginning with "rename" are used to rename interfaces. It takes one or more arguments in the form of "CUR=NEW", where CUR is the name of an existing interface, and NEW is the new name. This becomes very powerful when combined with pattern matching for the CUR interface.

Interfaces are renamed whenever "ifup" is called. Renaming logically happens before anything else is done. So if an interface is started with the name "foo", and it has to be renamed to "bar" and brought up at boot time, then one should use the following /etc/network/interfaces file:

```
rename foo=bar
auto bar
iface bar ...
```

However, if the interface is not renamed yet, it is possible to use both "ifup foo" and "ifup bar". The former command will then automatically be converted to the latter. This is mainly useful when ifup is called automatically whenever an interface is hotplugged.

Interface renaming only works if the operating system supports it, if an interface is not renamed to another existing interface, and may require that the interface that is to be renamed has not been brought up yet. If ifup tries to rename an interface and it fails, it will exit with an error.

INCLUDING OTHER FILES

Lines beginning with "source" are used to include stanzas from other files, so configuration can be split into many files. The word "source" is followed by the path of file to be sourced. Shell wildcards can be used. (See **wordexp**(3) for details.)

Similarly, "source-directory" keyword is used to source multiple files at once, without specifying them individually or using shell globs. Additionally, when "source-directory" is used, names of the files are checked to match the following regular expression: *^[a−zA−Z0−9_−]+$*. In other words, the names must consist entirely of ASCII upper- and lower-case letters, ASCII digits, ASCII underscores, and ASCII minus-hyphens. In the directory path, shell wildcards may be used as well.

When sourcing files or directories, if a path doesn't have a leading slash, it's considered relative to the directory containing the file in which the keyword is placed. In the example above, if the file is located at */etc/network/interfaces*, paths to the included files are understood to be under */etc/network*.

By default, on a freshly installed Debian system, the interfaces file includes a line to source files in the */etc/network/interfaces.d* directory.

MAPPINGS

Stanzas beginning with the word "mapping" are used to determine how a logical interface name is chosen for a physical interface that is to be brought up. The first line of a mapping stanza consists of the word "mapping" followed by a pattern in shell glob syntax. Each mapping stanza must contain a **script** definition. The named script is run with the physical interface name as its argument and with the contents of all following "map" lines (**without** the leading "map") in the stanza provided to it on its standard input. The script must print a string on its standard output before exiting. See */usr/share/doc/ifupdown/examples* for examples of what the script must print.

Mapping a name consists of searching the remaining mapping patterns and running the script corresponding to the first match; the script outputs the name to which the original is mapped.

ifup is normally given a physical interface name as its first non−option argument. **ifup** also uses this name as the initial logical name for the interface unless it is accompanied by a suffix of the form *=LOGICAL*, in which case ifup chooses *LOGICAL* as the initial logical name for the interface. It then maps this name, possibly more than once according to successive mapping specifications, until no further mappings are possible. If the resulting name is the name of some defined logical interface then **ifup** attempts to bring up the physical interface as that logical interface. Otherwise **ifup** exits with an error.

INTERFACE DEFINITIONS

Stanzas defining logical interfaces start with a line consisting of the word "iface" followed by the name of the logical interface. In simple configurations without mapping stanzas this name should simply be the name of the physical interface to which it is to be applied. (The default mapping script is, in effect, the

echo command.) The interface name is followed by the name of the address family that the interface uses. This will be "inet" for TCP/IP networking, but there is also some support for IPX networking ("ipx"), and IPv6 networking ("inet6"). Following that is the name of the method used to configure the interface.

Additional options can be given on subsequent lines in the stanza. Which options are available depends on the family and method, as described below. Additional options can be made available by other Debian packages. For example, the wireless−tools package makes available a number of options prefixed with "wireless−" which can be used to configure the interface using **iwconfig**(8)**.** (See **wireless**(7) for details.) A list of packages providing additional options is mentioned in the section "OPTIONS PROVIDED BY OTHER PACKAGE".

Options are usually indented for clarity (as in the example above) but are not required to be.

Multiple "iface" stanzas can be given for the same interface, in which case all of the configured addresses and options for that interface will be applied when bringing up that interface. This is useful to configure both IPv4 and IPv6 addresses on the same interface (although if no inet6 stanza is present, the kernel will normally still perform stateless address autoconfiguration if there is an IPv6 route advertisement daemon on the network). It can also be used to configure multiple addresses of the same type on a single interface.

INTERFACE TEMPLATES

It is possible to define interface definition templates and extend them using the **inherits** keyword:

```
iface ethernet inet static
      mtu 1500
      hwaddress 11:22:33:44:55:66

iface eth0 inet static inherits ethernet
      address 192.168.1.2/24
```

This may be useful to separate link-level settings shared by multiple interfaces from, for example, IP address settings specific to every interface.

PATTERN MATCHING INTERFACES

It is possible to use patterns to match one or more real interfaces. These patterns can currently appear in lines beginning with "auto", "allow-", "rename" and on the command line. A pattern has the following format (see below for exceptions for GNU/Hurd):

```
[VARIABLE]/VALUE[/[OPTIONS]][=LOGICAL]
```

If no VARIABLE is given, this pattern will match interface names against the given VALUE. VALUE can contain wildcard patterns such as ? and *, see the **fnmatch**(3) function. When **ifup** or **ifdown** is run, patterns are replaces by all real interfaces that are currently known to the operating system kernel and whose names match the pattern. For example, given the following line:

```
auto /eth*
```

If the kernel knows about the interfaces with names lo, eth0 and eth1, then the above line is then interpreted as:

```
auto eth0 eth1
```

Note that there must still be valid "iface" stanzas for each matching interface. However, it is possible to combine a pattern with a mapping to a logical interface, like so:

```
auto /eth*=eth
iface eth inet dhcp
```

Valid variable names are "mac", in which case value is matched against the interface's MAC address. On Linux, the variable name can also be any filename in /sys/class/net/<iface>/, in which case the value is matched against the contents of the corresponding file.

The OPTIONS field currently only supports a number. If given, only the n-th interface that has a matching value will actually be used, where n is the number given, starting at 1. So /eth*/1 will match the first interface whose name starts with eth.

On GNU/Hurd, interface names start with /dev/, and this obviously clashes with the format for patterns. To ensure an interface name like /dev/eth0 does not get interpreted as a pattern, any pattern that starts with /dev/ is ignored, and instead interpreted as a literal interface name. To make a pattern that matches interface names on GNU/Hurd, use something like:

```
auto /?dev?eth*=eth
iface eth inet dhcp
```

VLAN INTERFACES

To ease the configuration of VLAN interfaces, interfaces having **.** (full stop character) in the name are configured as 802.1q tagged virtual LAN interface. For example, interface **eth0.1** is a virtual interface with VLAN ID 1 having **eth0** as its parent interface.

VLAN interfaces are mostly treated as independent interfaces. As such, a VLAN interface is normally not automatically brought up when its parent interface is brought up. The exception is when ifup is called with the --allow option, in which case all VLAN interfaces that are in the same allow class as the parent interface are brought up together with the parent interface. For example:

```
allow-hotplug eth0 eth0.1

iface eth0 inet static
     address ...

iface eth0.1 inet static
     address ...

iface eth0.2 inet static
     address ...
```

In the above example, when "ifup --allow hotplug eth0" is called (either manually or because udev triggers this when a network device is hotplugged), the interface eth0 and the VLAN interface eth0.1 are brought up, but eth0.2 is not.

Keep in mind that pattern matching will only match interfaces the kernel knows about, so it is not possible to specify "auto /eth0.*" and have all VLAN interfaces for eth0 be brought up at boot time. Another way to ensure that a VLAN interface is brought up automatically when the parent interface is brought up, is to use a recursive call to ifup, like so:

```
iface eth0 inet manual
     up ifup eth0.3

iface eth0.3 inet static
     address ...
```

Note that there is no need to add an explicit call to ifdown, since VLAN interfaces are automatically brought down whenever their parent interfaces are brought down.

IFACE OPTIONS

The following "command" options are available for every family and method. Each of these options can be given multiple times in a single stanza, in which case the commands are executed in the order in which they appear in the stanza. (You can ensure a command never fails by suffixing them with "|| true".)

pre−up *command*
: Run *command* before bringing the interface up. If this command fails then **ifup** aborts, refraining from marking the interface as configured, prints an error message, and exits with status 0. This behavior may change in the future.

up *command*

post−up *command*
: Run *command* after bringing the interface up. If this command fails then **ifup** aborts, refraining from marking the interface as configured (even though it has really been configured), prints an

error message, and exits with status 0. This behavior may change in the future.

down *command*

pre−down *command*

Run *command* before taking the interface down. If this command fails then **ifdown** aborts, marks the interface as deconfigured (even though it has not really been deconfigured), and exits with status 0. This behavior may change in the future.

post−down *command*

Run *command* after taking the interface down. If this command fails then **ifdown** aborts, marks the interface as deconfigured, and exits with status 0. This behavior may change in the future.

description *name*

Alias interface by *name*

HOOK SCRIPTS

There are four directories in which scripts can be placed which will always be run for any interface during certain phases of ifup and ifdown commands. These are:

/etc/network/if-pre-up.d/

Scripts in this directory are run before bringing the interface up.

/etc/network/if-up.d/

Scripts in this directory are run after bringing the interface up.

/etc/network/if-down.d/

Scripts in this directory are run before bringing the interface down.

/etc/network/if-post-down.d/

Scripts in this directory are run after bringing the interface down.

The scripts in which are run (with no arguments) using **run−parts**(8) after the corresponding **pre-up**, **up**, **down** and **post-down** options in the */etc/network/interfaces* file itself have been processed. Please note that as **post−up** and **pre−down** are aliases, no files in the corresponding directories are processed. Please use *if-up.d* and *if-down.d* directories instead.

ENVIRONMENT VARIABLES

All hook scripts, and the commands executed by **pre-up**, **up**, **post-up**, **pre-down**, **down** and **post-down** have access to the following environment variables:

IFACE The physical name of the interface being processed, or "--all" (see below).

LOGICAL

The logical name of the interface being processed, or "auto" (see below).

ADDRFAM

The address family of the interface, or "meta" (see below).

METHOD

The method of the interface (e.g., *static*), or "none" (see below).

CLASS

The class of interfaces being processed. This is a copy of the value given to the **−−allow** option when running ifup or ifdown, otherwise it is set to "auto" when the **−−all** option is used.

CLASS

The class of interfaces being processed. This is a copy of the value given to the **−−allow** option when running ifup or ifdown, otherwise it is set to "auto" when the **−−all** option is used.

MODE

start if run from ifup, *stop* if run from ifdown.

PHASE

As per MODE, but with finer granularity, distinguishing the *pre-up*, *post-up*, *pre-down* and *post-down* phases.

VERBOSITY
Indicates whether **−−verbose** was used; set to 1 if so, 0 if not.

PATH The command search path: */usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin*

Additionally, all options given in an interface definition stanza are exported to the environment in upper case with "IF_" prepended and with hyphens converted to underscores and non−alphanumeric characters discarded.

When ifupdown is being called with the **−−all** option, before doing anything to interfaces, it calls all the hook scripts (*pre-up* or *down*) with **IFACE** set to "−−all", **LOGICAL** set to the current value of −−allow parameter (or "auto" if it's not set), **ADDRFAM**="meta" and **METHOD**="none". After all the interfaces have been brought up or taken down, the appropriate scripts (*up* or *post-down*) are executed.

CONCURRENCY AND PARALLEL EXECUTION

Ifupdown uses per-interface locking to ensure that concurrent ifup and ifdown calls to the same interface are run in serial. However, calls to different interfaces will be able to run in parallel. It is therefore important that any hook scripts and *pre-up*, *up*, *down* and *post-down* commands are written with the possibility of parallel execution in mind.

It is allowed to recursively call **ifup** and **ifdown** from hook scripts and interface commands, as long as these calls refer to a different interface than the one that is already being (de)configured. Loops are detected and will result in the call failing instead of a deadlock, although it is best if one does not rely on that.

OPTIONS PROVIDED BY OTHER PACKAGES

This manual page documents the configuration options provided by the ifupdown package. However, other packages can make other options available for use in /etc/network/interfaces. Here is a list of packages that provide such extensions:

arping, avahi-autoipd, avahi-daemon, bind9, bridge-utils, clamav-freshclam, controlaula, epoptes-client, ethtool, guidedog, hostap-utils, hostapd, htpdate, ifenslave, ifmetric, ifupdown-extra, ifupdown-multi, ifupdown-scripts-zg2, initscripts, isatapd, linux-wlan-ng, lprng, macchanger, miredo, nslcd, ntpdate, openntpd, openresolv, openssh-server, openvpn, openvswitch-switch, postfix, resolvconf, sendmail-base, shorewall-init, slrn, slrnpull, tinc, ucarp, uml-utilities, uruk, vde2, vlan, vzctl, whereami, wide-dhcpv6-client, wireless-tools, wpasupplicant.

Please consult the documentation of those packages for information about how they extend ifupdown.

INET ADDRESS FAMILY

This section documents the methods available in the inet address family.

The loopback Method

This method may be used to define the IPv4 loopback interface.

Options

(No options)

The static Method

This method may be used to define Ethernet interfaces with statically allocated IPv4 addresses.

Options

address *address*
Address (dotted quad/netmask) **required**

netmask *mask*
Netmask (dotted quad or number of bits) **deprecated**

broadcast *broadcast_address*
Broadcast address (dotted quad, + or -) **deprecated**. Default value: "+"

metric *metric*
Routing metric for default gateway (integer)

gateway *address*
Default gateway (dotted quad)

pointopoint *address*
Address of other end point (dotted quad). Note the spelling of "point-to".

hwaddress *address*
Link local address or "random".

mtu *size*
MTU size

scope Address validity scope. Possible values: global, link, host

The manual Method

This method may be used to define interfaces for which no configuration is done by default. Such interfaces can be configured manually by means of **up** and **down** commands or /etc/network/if-*.d scripts.

Options

hwaddress *address*
Link local address or "random".

mtu *size*
MTU size

The dhcp Method

This method may be used to obtain an address via DHCP with any of the tools: dhclient, pump, udhcpc, dhcpcd. (They have been listed in their order of precedence.) If you have a complicated DHCP setup you should note that some of these clients use their own configuration files and do not obtain their configuration information via **ifup**.

Options

hostname *hostname*
Hostname to be requested (pump, dhcpcd, udhcpc)

metric *metric*
Metric for added routes (dhclient)

leasehours *leasehours*
Preferred lease time in hours (pump)

leasetime *leasetime*
Preferred lease time in seconds (dhcpcd)

vendor *vendor*
Vendor class identifier (dhcpcd)

client *client*
Client identifier (dhcpcd)

hwaddress *address*
Hardware address.

The bootp Method

This method may be used to obtain an address via bootp.

Options

bootfile *file*
Tell the server to use *file* as the bootfile.

server *address*
Use the IP address *address* to communicate with the server.

hwaddr *addr*
Use *addr* as the hardware address instead of whatever it really is.

The tunnel Method

This method is used to create GRE or IPIP tunnels. You need to have the **ip** binary from the **iproute** package. For GRE tunnels, you will need to load the ip_gre module and the ipip module for IPIP tunnels.

Options

address *address*
Local address (dotted quad) **required**

mode *type*
Tunnel type (either GRE or IPIP) **required**

endpoint *address*
Address of other tunnel endpoint **required**

dstaddr *address*
Remote address (remote address inside tunnel)

local *address*
Address of the local endpoint

metric *metric*
Routing metric for default gateway (integer)

gateway *address*
Default gateway

ttl *time* TTL setting

mtu *size*
MTU size

The ppp Method

This method uses pon/poff to configure a PPP interface. See those commands for details.

Options

provider *name*
Use *name* as the provider (from /etc/ppp/peers).

unit *number*
Use *number* as the ppp unit number.

options *string*
Pass *string* as additional options to pon.

The wvdial Method

This method uses wvdial to configure a PPP interface. See that command for more details.

Options

provider *name*
Use *name* as the provider (from /etc/wvdial.conf).

The ipv4ll Method

This method uses avahi-autoipd to configure an interface with an IPv4 Link-Layer address (169.254.0.0/16 family). This method is also known as APIPA or IPAC, and often colloquially referred to as "Zeroconf address".

Options

(No options)

IPX ADDRESS FAMILY

This section documents the methods available in the ipx address family.

The static Method

This method may be used to setup an IPX interface. It requires the *ipx_interface* command.

Options

frame *type*
type of Ethernet frames to use (e.g. **802.2**)

netnum *id*
Network number

The dynamic Method

This method may be used to setup an IPX interface dynamically.

Options

frame *type*
type of Ethernet frames to use (e.g. **802.2**)

INET6 ADDRESS FAMILY

This section documents the methods available in the inet6 address family.

The auto Method

This method may be used to define interfaces with automatically assigned IPv6 addresses. Using this method on its own doesn't mean that RDNSS options will be applied, too. To make this happen, **rdnssd** daemon must be installed, properly configured and running. If stateless DHCPv6 support is turned on, then additional network configuration parameters such as DNS and NTP servers will be retrieved from a DHCP server. Please note that on ifdown, the lease is not currently released (a known bug).

Options

privext *int*
Privacy extensions (RFC4941) (0=off, 1=assign, 2=prefer)

accept_ra *int*
Accept router advertisements (0=off, 1=on, 2=on+forwarding). Default value: "2"

dhcp *int*
Use stateless DHCPv6 (0=off, 1=on)

request_prefix *int*
Request a prefix through DHCPv6 Prefix Delegation (0=off, 1=on). Default value: "0"

ll-attempts
Number of attempts to wait for a link-local address. Default value: "60"

ll-interval
Link-local address polling interval in seconds. Default value: "0.1"

The loopback Method

This method may be used to define the IPv6 loopback interface.

Options

(No options)

The static Method

This method may be used to define interfaces with statically assigned IPv6 addresses. By default, stateless autoconfiguration is disabled for this interface.

Options

address *address*
Address (colon delimited/netmask) **required**

netmask *mask*
: Netmask (number of bits, eg 64) **deprecated**

metric *metric*
: Routing metric for default gateway (integer)

gateway *address*
: Default gateway (colon delimited)

media *type*
: Medium type, driver dependent

hwaddress *address*
: Hardware address or "random"

mtu *size*
: MTU size

accept_ra *int*
: Accept router advertisements (0=off, 1=on, 2=on+forwarding)

autoconf *int*
: Perform stateless autoconfiguration (0=off, 1=on). Default value: "0"

privext *int*
: Privacy extensions (RFC3041) (0=off, 1=assign, 2=prefer)

scope
: Address validity scope. Possible values: global, site, link, host

preferred-lifetime *int*
: Time that address remains preferred

dad-attempts
: Number of attempts to settle DAD (0 to disable DAD). Default value: "60"

dad-interval
: DAD state polling interval in seconds. Default value: "0.1"

The manual Method

This method may be used to define interfaces for which no configuration is done by default. Such interfaces can be configured manually by means of **up** and **down** commands or /etc/network/if-*.d scripts.

Options

hwaddress *address*
: Hardware address or "random"

mtu *size*
: MTU size

The dhcp Method

This method may be used to obtain network interface configuration via stateful DHCPv6 with dhclient. In stateful DHCPv6, the DHCP server is responsible for assigning addresses to clients.

Options

hwaddress *address*
: Hardware address or "random"

accept_ra *int*
: Accept router advertisements (0=off, 1=on, 2=on+forwarding). Default value: "1"

autoconf *int*
: Perform stateless autoconfiguration (0=off, 1=on)

request_prefix *int*
: Request a prefix through DHCPv6 Prefix Delegation (0=off, 1=on). Default value: "0"

ll-attempts
Number of attempts to wait for a link-local address. Default value: "60"

ll-interval
Link-local address polling interval in seconds. Default value: "0.1"

The v4tunnel Method

This method may be used to setup an IPv6-over-IPv4 tunnel. It requires the **ip** command from the **iproute** package.

Options

address *address*
Address (colon delimited/netmask) **required**

netmask *mask*
Netmask (number of bits, eg 64) **deprecated**

endpoint *address*
Address of other tunnel endpoint (IPv4 dotted quad) **required**

local *address*
Address of the local endpoint (IPv4 dotted quad)

metric *metric*
Routing metric for default gateway (integer)

gateway *address*
Default gateway (colon delimited)

ttl *time* TTL setting

mtu *size*
MTU size

preferred-lifetime *int*
Time that address remains preferred

The 6to4 Method

This method may be used to setup an 6to4 tunnel. It requires the **ip** command from the **iproute** package.

Options

local *address*
Address of the local endpoint (IPv4 dotted quad) **required**

metric *metric*
Routing metric for default gateway (integer)

ttl *time* TTL setting

mtu *size*
MTU size

preferred-lifetime *int*
Time that address remains preferred

CAN ADDRESS FAMILY

This section documents the methods available in the can address family.

The static Method

This method may be used to setup an Controller Area Network (CAN) interface. It requires the the **ip** command from the **iproute** package.

Options

bitrate *bitrate*
: bitrate (1..1000000) **required**

samplepoint *samplepoint*
: sample point (0.000..0.999)

loopback *loopback*
: loop back CAN Messages (on|off)

listenonly *listenonly*
: listen only mode (on|off)

triple *triple*
: activate triple sampling (on|off)

oneshot *oneshot*
: one shot mode (on|off)

berr *berr*
: activate berr reporting (on|off)

KNOWN BUGS/LIMITATIONS

The **ifup** and **ifdown** programs work with so-called "physical" interface names. These names are assigned to hardware by the kernel. Unfortunately it can happen that the kernel assigns different physical interface names to the same hardware at different times; for example, what was called "eth0" last time you booted is now called "eth1" and vice versa. This creates a problem if you want to configure the interfaces appropriately. A way to deal with this problem is to use mapping scripts that choose logical interface names according to the properties of the interface hardware. See the **get-mac-address.sh** script in the examples directory for an example of such a mapping script. See also Debian bug #101728.

AUTHOR

The ifupdown suite was written by Anthony Towns <aj@azure.humbug.org.au>. This manpage was contributed by Joey Hess <joey@kitenet.net>.

SEE ALSO

ifup(8), **ip**(8), **ifconfig**(8), **run−parts**(8), **resolvconf**(8).

For advice on configuring this package read the **Network Configuration** chapter of the *Debian Reference* manual, available at *http://www.debian.org/doc/manuals/debian-reference/ch05.en.html* or in the **debian-reference-en** package.

Examples of how to set up interfaces can be found in **/usr/share/doc/ifupdown/examples/network-interfaces.gz**.