Translation(s): English - Español - Français - Italiano - 简体中文

Reader Prerequisites: To get the most from this article, understand the following concepts before reading: basic unix command line tools, text editors, DNS, TCP/IP, DHCP, netmask, gateway

Table of Contents

PomocSpisTreści

Setting up an Ethernet Interface

The majority of network setup can be done via the **`interfaces`** configuration file at **`/etc/network/interfaces`**. Here, you can give your network card an IP address (or use dhcp), set up routing information, configure IP masquerading, set default routes and much more.

Remember to add interfaces that you want brought up at boot time to the 'auto' line.

See **`man interfaces`** for more options.

Using DHCP to automatically configure the interface

If you're just using DHCP then all you need is something like:

```
auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
```

For DHCPv6 (used for IPv6), **append** also the following `iface` stanza

```
iface eth0 inet6 dhcp
```

Alternatively, IPv6 can be autoconfigured using *stateless address autoconfiguration,* or SLAAC, which is specified using `auto` instead of `dhcp` in the `inet6` stanza:

```
iface eth0 inet6 auto
```

Configuring the interface manually

If you're configuring it manually then something like this will set the default gateway (network, broadcast and gateway are optional):

```
auto eth0
iface eth0 inet static
    address 192.0.2.7
    netmask 255.255.255.0
    gateway 192.0.2.254
```

If you want to add an IPv6 address, too, **append** something like:

```
iface eth0 inet6 static
    address 2001:db8::c0ca:1eaf
    netmask 64
    gateway 2001:db8::1ead:ed:beef
```

See **`man interfaces`** for more options.

Setting the speed and duplex

Autonegotiation repeatedly failing is often a symptom of faulty cabling, so investigate physical matters before assuming that the interfaces' autonegotiation algorithms are incompatible. If you turn off autonegotiation and set speed and duplex manually then the partner interface at the other

end of the cable will assume that the absence of autonegotiation indicates a speed of 10Mbps and a duplex of half. For error-free operation if you set speed and duplex manually you must ensure that exactly the same speed and duplex are configured on the partner interface.

If you set your interface's speed and duplex by hand, then some trial and error may be required. Here are the basic steps:

- Install the DebianPkg: ethtool and DebianPkg: net-tools packages, so that you have the `ethtool` and `mii-tool` programs. One or both of these might work for your interface.
- Make sure you have a way to login to the system in case the network interface becomes nonfunctional. An SSH connection could be disrupted, so you should have a fallback strategy.
- Identify the interface in question (it will often be eth0). Adjust the remainder of these instructions accordingly.
- Try to determine what its current speed and duplex settings are. This is where it gets fun:
  - As root, try `ethtool eth0` first, and see whether the "Speed:" and "Duplex:" lines look valid. If not, the `ethtool` may not be supported by your device.
  - As root, try `mii-tool -v eth0` and see whether its output looks correct. If not, them `mii-tool` may not be supported by your device.
  - If neither one is supported, you may have to set parameters directly on the kernel driver module. Identify which driver module you're using by reading the output of `dmesg` and `lsmod`. You can then try `modinfo MODULENAME` to see what parameters it accepts, if any. (You can use `modinfo` even on modules that are not loaded, for comparison.) *ToDo: where does*

one set kernel module parameters?

- Next, try to change the settings of the interface while it's operating. You'll need to be root, of course. Either:
  - `ethtool -s eth0 speed 100 duplex full autoneg off` (assuming 100 Mbps and full duplex)
  - `mii-tool -F 100baseTx-FD eth0` (same assumption)

  In each case, re-check to see whether the interface settings actually changed, and then try sending some data in and out of the system to see whether the NIC is operating correctly.
- If one of these commands successfully set your NIC, then you can put it into `/etc/network/interfaces` so it runs when you bring the interface up (e.g. at boot time). However, before you do that, you should understand that some drivers and devices behave differently than others. When the driver module is loaded, the NIC may begin autonegotiation without any way to stop it (particularly with drivers that do not accept parameters). The settings from `interfaces` are applied at some point after that, which may be right in the middle of the negotiation. So, some people find it necessary to delay the `ethtool` or `mii-tool` command by a few seconds. Thus:

```
iface eth0 inet static
       address ...
       netmask ...
       gateway ...
       up sleep 5; ethtool -s eth0 ...
```

  Or the analogous `mii-tool` command.
- Reboot the machine to make sure it comes up correctly, and be prepared to intervene manually (e.g. Ctrl-Alt-Del and then boot into single-user mode from GRUB or LILO) if

things don't work.

Bringing up an interface without an IP address

To create a network interface without an IP address at all use the manual method and use pre-up and post-down commands to bring the interface up and down.

```
iface eth0 inet manual
   pre-up ifconfig $IFACE up
   post-down ifconfig $IFACE down
```

If the interface is a VLAN interface, the up/down commands must be executed after/before the vlan hooks. (You also have to install the DebianPkg: vlan package.)

```
iface eth0.99 inet manual
   post-up ifconfig $IFACE up
   pre-down ifconfig $IFACE down
```

Note: If you create the VLAN interface only to put it into a bridge, there is no need to define the VLAN interface manually. Just config the bridge, and the VLAN interface will be created automatically when creating the bridge (see below).

Defining the (DNS) Nameservers

Before a computer can connect to an external network resource (say, for example, a web server), it must have a means of converting any alpha-numeric names (e.g.

wiki.debian.org) into numeric network addresses (e.g. 140.211.166.4). (The Internet uses these structured numeric IP addresses as network addresses.)

The C library and other resolver libraries look to `/etc/resolv.conf` for a list of nameservers. In the simplest case, that is the file to edit to set the list of name servers. But note that various other programs for dynamic configuration will be happy to overwrite your settings:

1. The **`resolvconf`** program
2. The **`network-manager`** daemon
3. DHCP clients

In most situations, the file to edit is the configuration file for such a program.

In the most complex situations, using **`resolvconf`** really is the way to go, though in more simple configurations it is probably overkill.

The resolv.conf configuration file

The configuration file **`resolv.conf`** at `/etc/resolv.conf` contains information that allows a computer connected to a network to resolve names into addresses. (Note: Do not confuse this *configuration file* with the *program* **`resolvconf`**, which unfortunately has a nearly identical name.)

The **`resolv.conf`** file typically contains the IP addresses of nameservers (DNS name resolvers) that will attempt to translate names into addresses for any node available on the network. There will be a line or lines that look like this:

```
nameserver 12.34.56.78
```

```
nameserver 12.34.56.79
```

In this example, the system is using nameservers at the IP addresses `12.34.56.78` and `12.34.56.79`. Simply edit the file and enter the IP addresses of the nameservers you need to use after each `nameserver`. Add more `nameserver` lines if you have more nameservers. **Don't use this method if you have the** `resolvconf` **program installed.**

The **`resolv.conf`** configuration file has many other options for defining how resolver looks up names. See **`man resolv.conf`** for details.

The resolvconf program

The **`resolvconf`** program keeps track of system information about the currently available nameservers. It should not be confused with the *configuration file* **`resolv.conf`**, which unfortunately has a nearly identical name. The **`resolvconf`** program is optional on a Debian system.

The configuration file **`resolv.conf`** contains information about the the nameservers to be used by the system. However, when multiple programs need to dynamically modify the **`resolv.conf`** configuration file they can step on each other and the file can become out-of-sync. The **`resolvconf`** program addresses this problem. It acts as an intermediary between programs that supply nameserver information (e.g. dhcp clients) and programs that use nameserver information (e.g. resolver).

When **`resolvconf`** is properly installed, the **`resolv.conf`** configuration file at `/etc/resolv.conf` is replaced by a symbolic link to `/etc/resolvconf/run/resolv.conf` and the resolver instead uses the configuration file that is

dynamically generated by **resolvconf** at `/etc/resolvconf/run/resolv.conf`.

The **resolvconf** program is generally only necessary when a system has multiple programs that need to dynamically modify the nameserver information. In a simple system where the nameservers do not change often or are only changed by one program, the **resolv.conf** configuration file is adequate.

If the **resolvconf** program is installed, you should not edit the **resolv.conf** configuration file manually as it will be dynamically changed by programs in the system. If you need to manually define the nameservers (as with a static interface), add a line something like the following to the **interfaces** configuration file at `/etc/network/interfaces`:

```
dns-nameservers 12.34.56.78 12.34.56.79
```

Place the line indented within an `iface` stanza, e.g., right after the `gateway` line. Enter the IP addresses of the nameservers you need to use after `dns-nameservers`. Put all of them on one line separated by spaces. Don't forget the "`s`" on the end of `dns-nameservers`.

The **resolvconf** program is a fairly new addition to Debian and many older programs need to be updated or reconfigured to work properly with it. If you have problems, see `/usr/share/doc/resolvconf/README`. It has lots of information on making other programs get along with **resolvconf**.

DNS configuration for NetworkManager

NetworkManager will override dhcp settings, overwriting

resolv.conf even if you've configured DNS in /etc/dhcp /dhclient.conf, e.g. causing DNS to first search the local domain, which may have to time out before DNS resolution continues causing lengthy DNS resolution times. You can get an idea of what NetworkManager thinks the settings should be by executing nm-tool at the command line.

You may configure these settings using a GUI by launching nm-connection-editor which currently (13.11.23) isn't to be found in System Tools → Administration menu, rather it must be launched by hand from the command line. After launching:

1. Choose a connection (from the Wired or Wireless tab) and click Edit.
2. Click on the IPv4 Settings tab
3. Choose 'Automatic (DHCP) addresses only' instead of just 'Automatic (DHCP)'.
4. Enter the DNS servers in the “DNS servers” field, separated by spaces (e.g. 208.67.222.222 for OpenDNS).
5. Click “Apply.”

NetworkManager saves these settings in `/etc/NetworkManager/system-connections/name-of-connection`. Example `/etc/NetworkManager/system-connections/Wired connection 1` :

```
[802-3-ethernet]
duplex=full
mac-address=XX:XX:XX:XX:XX:XX

[connection]
id=Wired connection 1
```

```
uuid=xxx-xxxxxx-xxxxxx-xxxxxx-xxx
type=802-3-ethernet
timestamp=1385213042

[ipv6]
method=auto

[ipv4]
method=auto
dns=208.67.222.222;
ignore-auto-dns=true
```

Running nm-tool again should show that NetworkManager now has the right idea of how your DNS should be resolved.

DHCP Client Configuration

Example: dhclient3 uses `/etc/dhcp/dhclient.conf`. The setting you want is

```
supersede domain-name-servers 12.34.56.78, 12.34.56.79
```

or perhaps

```
prepend domain-name-servers 12.34.56.78, 12.34.56.79;
```

See the dhclient.conf(5) manual page for details.

Bridging

Bridging puts multiple interfaces into the same network

segment. This is very popular when connecting a server to multiple switches for high availability or with virtualization. In the latter case it is usually used to create a bridge in the host (eg. dom0) and put the virtual interfaces of the guests (domU) into the bridge.

Example: Connect a server to 2 switches (via eth0 and eth1) by defining bridge 0 and give the server an IP address in this subnet:

```
auto br0
iface br0 inet static
        address 10.10.0.15
        netmask 255.255.255.0
        gateway 10.10.0.1
        bridge_ports eth0 eth1
        up /usr/sbin/brctl stp br0 on
```

If a server is connected to multiple switches then you usually need to run the spanning tree protocol to avoid loops. Therefore STP must be turned on via an "up" command as shown above.

Example: Bridge setup without IP address configuration (use "manual" instead of "static") to "forward" an interface to a guest VM. (The static bridge config contains only 1 physical interface. The virtual interface will be added to the bridge when the VM is started.)

```
auto br1
iface br1 inet manual
        bridge_ports eth4
```

```
        up /usr/sbin/brctl setageing br1 0
        up /usr/sbin/brctl stp br0 off
```

Note: The Linux bridge supports only STP, no RSTP (Rapid Spanning Tree). Therefore it supports only the old STP Costs, not the new RSTP Costs (see WikiPedia: Spanning_Tree_Protocol). This is usually fine with Cisco Switches, but eg. Juniper switches use the RSTP costs and therefore this may lead to different spanning tree calculations and loop problems. This can be fixed by settings the costs manually, either on the switch or on the server. Setting the cost on the switch is preferred as Linux switches back to the default costs whenever an interface does down/up.

Bridging without Switching

By default the Linux bridge acts like a switch. This means, it remembers the MAC addresses behind a switch port and if the destination MAC address is known, data packets or only forward to the respective port - otherwise packets will be broadcasted.

In some setups this is bad. For example if the bridge connects 2 trunk interfaces and the same MAC addresses may be seen from both interfaces, depending on the VLAN. As the Linux bridge does not support VLANs (dedicated MAC address tables per each VLAN), in such setups you have to disable the MAC address learning and put the bridge into a real "bridge" mode with:

```
        up /sbin/brctl setageing br0 0
        up /sbin/brctl stp br0 off
```

Howto use vlan (dot1q, 802.1q, trunk) (Etch, Lenny)

Manual config

```
modprobe 8021q

apt-get install vlan

vconfig add eth0 222    # 222 is vlan number
ifconfig eth0.222 up
ifconfig eth0.222 mtu 1496    #optional if your networ
ifconfig eth0.222 10.10.10.1 netmask 255.255.255.0
```

Network init script config

Into /etc/modules add line:

```
8021q
```

In /etc/network/interfaces to section iface add parameter:

```
        vlan-raw-device eth0
```

The interface name should be the raw interface name (the same as specified by vlan-raw-device), then a dot, then the VLAN ID, for example eth0.100. It can instead be "vlan" then the VLAN ID, for example vlan100. In either case, the VLAN ID is on the end, and this is the only place that it is configured.

Note: If you name your VLAN interfaces ethX.YYY, then there is no need to specify the vlan-raw-device, as the raw device can be retrieved from the interface name.

Eg:

```
auto eth0.222
iface eth0.222 inet static
        address 10.10.10.1
        netmask 255.255.255.0
        vlan-raw-device eth0
```

Bridges and VLANs

If you create VLAN interfaces only to put them into a bridge, there is no need to define the VLAN interfaces manually. Just config the bridge, and the VLAN interface will be created automatically when creating the bridge, e.g:

```
auto br1
iface br1 inet manual
        bridge_ports eth0.99 eth1.99
        up /usr/sbin/brctl stp br0 on
```

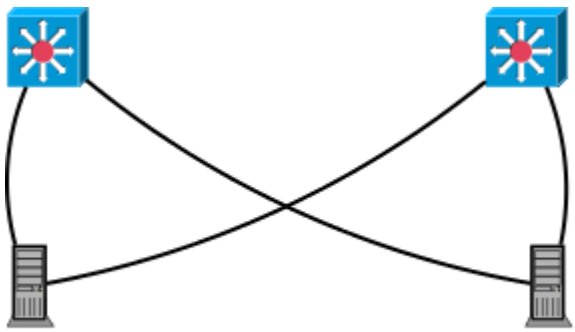
Caveats when using bridging and vlan

```
#/etc/network/interfaces
auto eth0 bri0
iface eth0 inet static
        address 192.168.1.1
```

```
        netmask 255.255.255.0
iface eth0.110 inet manual
        vlan-raw-device eth0
iface bri0 inet static
        address 192.168.110.1
        netmask 255.255.255.0
        bridge_ports eth0.110
        bridge_stp on
        bridge_maxwait 10
```

If you are using a brigded VLAN setup, which is probably useful for networking in virtualization environments, take care to only attach either a bridge device or VLAN devices to an underlying physical device - like shown above. Attaching the physical interface (eth0) to a bridge (eg. bri1) while using the same physical interface on apparently different VLANs will result in all packets to remain tagged. (Kernel newer than 2.6.37 and older than 3.2).

Howto create fault tolerant bonding with vlan (Etch)



debian_bonding.dia

How to configure one of the above server active backup

bonding 3 vlan {vlan10,vlan20,vlan30} Debian networking without SPOF without native vlan.

```
aptitude install vlan ifenslave-2.6
```

Network config

Cisco switch interface example config

```
interface GigabitEthernet1/2
 description eth1
 switchport
 switchport trunk encapsulation dot1q
 switchport trunk allowed vlan 10,20,30
 switchport mode trunk
 no ip address
 no cdp enable
 spanning-tree portfast trunk
```

bonding with active backup

Create a file `/etc/modprobe.d/bonding.conf` containing:

```
alias bond0 bonding
options bonding mode=active-backup miimon=100 downdela
```

/etc/network/interfaces

```
# The loopback network interface
```

```
auto lo
iface lo inet loopback
# The primary network interface
auto bond0
iface bond0 inet manual
        up ifconfig bond0 0.0.0.0 up
        slaves eth1 eth0
auto vlan10
iface vlan10 inet static
        address 10.10.10.12
        netmask 255.255.0.0
        vlan-raw-device bond0
        gateway 10.10.0.1
        dns-search hup.hu
        dns-nameservers 10.10.0.2
auto vlan20
iface vlan20 inet static
        address 10.20.10.12
        netmask 255.255.0.0
        vlan-raw-device bond0
auto vlan30
iface vlan30 inet static
        address 10.30.10.12
        netmask 255.255.0.0
        vlan-raw-device bond0
```

Multiple IP addresses on One Interface

Interface aliasing allows one interface to have multiple IP addresses. This is useful when more than one server is to be visible *via* the Internet. Note that virtual hosts can support multiple Apache servers with a single IP address. Apache

responds to the domain name supplied by the client in the HTTP header. In many other situations, one external IP is needed for each server using a port.

Legacy method

This /etc/network/interfaces text assigns three IP addresses to eth0.

```
auto eth0
allow-hotplug eth0
iface eth0 inet static
    address 192.168.1.42
    netmask 255.255.255.0
    gateway 192.168.1.1

auto eth0:0
allow-hotplug eth0:0
iface eth0:0 inet static
    address 192.168.1.43
    netmask 255.255.255.0

auto eth0:1
allow-hotplug eth0:1
iface eth0:1 inet static
    address 192.168.1.44
    netmask 255.255.255.0
```

An alias interface should not have "gateway" or "dns-nameservers"; dynamic IP assignment is permissible.

The above configuration is the previous traditional method that reflects the traditional use of *ifconfig* to configure

network devices. *ifconfig* has introduced the concept of *aliased* or *virtual* interfaces. Those types of virtual interfaces have names of the form *interface:integer* and ifconfig treats them very similarly to real interfaces.

Nowadays *ifupdown* uses *ip* utility from the *iproute2* package instead of *ifconfig*. The newer *ip* utility does not use the same concept of aliases or virtual interfaces. However, it supports assigning arbitrary names to the interfaces (they're called labels). *ifupdown* uses this feature to support aliased interfaces while using *ip*.

iproute2 method

Also, *ifupdown* supports specifying multiple interfaces by repeating *iface* sections with the same interface name. The key difference from the method described above is that all such sections are treated by *ifupdown* as just one interface, so user can't add or remove them individually. However, *up*/*down* commands, as well as scripts, are called for every section as it used to be.

Note however that this method is **dangerous**! Certain driver/hardware combinations may sometimes fail to bring the link up if no labels are assigned to the alias interfaces. (Seen this on Debian Wheezy and Jessie with RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller (rev 01) auto-negotiating to 10/full. A similar warning from another person exists in the history of this page.)

This /etc/network/interfaces text assigns three IP addresses to eth0.

```
auto eth0
allow-hotplug eth0
```

```
iface eth0 inet static
    address 192.168.1.42
    netmask 255.255.255.0
    gateway 192.168.1.1

iface eth0 inet static
    address 192.168.1.43
    netmask 255.255.255.0

iface eth0 inet static
    address 192.168.1.44
    netmask 255.255.255.0

# adding IP addresses from different subnets is also po
iface eth0 inet static
    address 10.10.10.14
    netmask 255.255.255.0
```

Manual approach:

```
auto eth0
allow-hotplug eth0
iface eth0 inet static
    address 192.168.1.42
    netmask 255.255.255.0
    gateway 192.168.1.1
    up   ip addr add 192.168.1.43/24 dev $IFACE label $
    down ip addr del 192.168.1.43/24 dev $IFACE label $
    up   ip addr add 192.168.1.44/24 dev $IFACE label $
    down ip addr del 192.168.1.44/24 dev $IFACE label $
    up   ip addr add 10.10.10.14/24 dev $IFACE label $I
```

```
    down ip addr del 10.10.10.14/24 dev $IFACE label $
```

CategoryNetwork