Linux Networking Guide: Part 1

CREATING A CONNECTION

Bruce Richardson presents the first part of our guide to configuring Linux networks from the command line. In this issue we look at how to connect a computer running Linux to a network

Whatever the virtues of the applications that can be run on Linux, one of its greatest strengths is its power and flexibility as a network operating system. Some people (including me) believe that you haven't really done grown up computing until you've done networked computing.

This series of articles is designed to show how Linux networks can be configured using the simple command line tools and text configuration files common to all Linux systems, using tools that are common to kernels from 2.0.x to 2.4.x. Where of interest, examples of configuration from specific distributions will be given but the emphasis throughout will be on portability. The article will restrict itself to IPv4 only.

This article describes how to choose and install a network card and how to configure and activate a network interface. Future articles will build on this to show how complex networks can be build from simple tools.

Network essentials

To function as part of a network your computer needs, as a minimum:

- A physical connection to the network. There are quite a few ways to connect to a modern network but this article will concentrate on the most common way: the internal Network Interface Card, henceforth referred to as a NIC.
- An address on the network. It's no use being able to talk to other computers if they can't find a return address to talk back to.
- A way of determining how to reach any given address. That is to say, given an arbitrary address to connect to, can we find it on this network, can we reach it through this network or must we find some other route?

This article covers items one and two. The next article will use item three as the basis for a demonstration of how a Linux box can be turned into a router and gateway for a network.

The physical connection

There are several ways to connect a PC to a network but we're going to look at internal network cards. If you're doing this on a budget, maybe with second hand kit inherited from a kind benefactor, choice may not come into it. If you do have a choice then there are several things to consider:

Price You get what you pay for. Cheap generic cards can be found for as little as £12 but don't expect them to perform well at high loads or to auto-negotiate connections reliably (if at all). It may also be difficult to identify the chipset on a generic card, which will make it difficult to find the correct driver. Personally, I would be sure to buy a reliable, well-known brand and I wouldn't consider spending less than £30 on a card.

Network type What kind of network do you want to connect to or set-up? A BNC/Coax network or RJ45/Cat5 network, 10Mbps or 100Mbps (See the

BNC versus RJ45

Older Ethernet networks (and cheap home network starter kits) use cards with BNC connectors, connected together with coaxial cable. BNC NICs can be identified by the stubby metal cylinder that is the connector. This kind of set-up is slow (maximum theoretical speed of 10Mbps) and fiddly (the cable must be terminated at each end, you may need to cut and crimp the cable yourself). However, it does have the advantage that you can connect more than two computers using just one cable: you simply add T-connectors to the cable at appropriate intervals.

Newer networks use cards with RJ45 connectors, connected together with UTP cable. RJ45 nics have sockets which look similar to those on modems. The cards typically come in 10Mbps or 100Mbps (Fast Ethernet) speeds, 100Mbps being the standard at the time of writing (Gigabit Ethernet is mostly restricted to the core of high-performance networks). This kind of network is faster but a UTP cable can only be used to connect two devices. If you want more than two computers on your network you will need to connect them to a hub or switch.

It is possible to get combo cards, which have both BNC and RJ45 connectors, though I've never seen one that was faster than 10Mbps.

BNC versus RJ45 boxout)? If you have the option, choose a Fast Ethernet set-up. It's the current standard for connecting workstations and the equipment will be easier to configure and troubleshoot.

Support Obviously you want a card that is supported by Linux but you should also try and find out how well the card is supported, whether the driver is stable or experimental etc. In this case the Internet is your friend. Find any identifying information about your network card (the make and model for reference, failing that the FCC ID number, which should be printed on the card somewhere, failing that any identifiable numbers on the card). Go to the Google Linux search and type in the information. You should find plenty of information to help make your decision. You may find it useful to cross-check whatever you find with the networking documentation accompanying the kernel source, which can normally be found in */usr/src/linux/documentation/networking*.

Installing and testing the card

I'm going to assume you know how to install an extension card into a PC. Once it is in we can do some basic checks to see if the hardware is functioning properly. This is important: there's no point going on to later steps if the card isn't functioning, you'll only waste a lot of time.

Start the machine and have a look at the card. Any decent, recent card will have at least one status light, one of which will show if any power is getting to the card (another reason to avoid cheap or old kit).

If your card seems to be getting power, get a cable and connect the card to another networking device, preferably one that has a connection status light (e.g. a hub or a good quality card in another PC). Don't forget to use the right kind of cable for the connection (see the Plugging your network together boxout) and be sure that the devices match (i.e. both operate at the same speed or at least one is capable of auto-negotiation). A connection light should show on one or both devices.

If you get no power light then either you need to reseat the card in its slot or the card is broken. If you get a power light but no connection light then you may have the wrong cable, one/both of the connected devices may not be functioning or the two devices may be mismatched (i.e. they are at operating at different speeds and the fast one isn't auto-negotiating): try changing the cable or the device you connect to until you get a working connection. If you can't get a connection light after trying several different cables and devices then your card is probably broken. When troubleshooting a problem connection, it's important to change one thing at a time. That way, if you do finally get a connection, you will be able to identify the problem component conclusively.

Plugging your network together

You can build a network out of all kinds of different parts. An IP packet doesn't care about the type of cable it travels across or its speed. On your network you might have a 10Mbps link from your PC to a hub, a 100Mbps link from the hub to a switch further up the line and a Gigabit connection between the switch and the application servers.

The physical connections are not hard to set up, since most modern networking kit automatically senses the speed of the device at the other end of the cable and will slow down to match slower kit (this is called auto-negotiation). There are even hubs with both RJ45 and BNC connectors. Do remember to use the right kind of cable. For 100Mbps speeds the UTP cable must be Cat 5 quality or higher. Cat 3 UTP is enough for 10Mbps but means you'd have to rewire before upgrading.

The simplest type of network is formed by linking two computers together. If using a UTP cable you will need crossover cable rather than a standard UTP patch cable. You can also connect multiple computers through a hub or switch and can chain hubs and/or switches together to create as large a network as you like. Use crossover cable to chain hubs and switches but standard patch cable to connect PCs to hubs or to switches.

The software connection

Before we can begin configuring the network interface we first have to make sure that we are using the right driver. Now, I absolutely recommend that you load NIC drivers as modules. It makes troubleshooting and testing so much easier:

- You can load and unload different modules, or the same modules with different parameters, as many times as you like with no need for a reboot.
- There is no need to recompile the kernel. Even if the module(s) you need (or want to test) aren't in your current set, you can compile them separately from the kernel (though you will need the config used to compile the kernel). If the driver is compiled into the kernel and you want to try a different driver or change the parameters passed to the driver then you will need at least to reboot and (in the former case) to recompile the kernel as well. This can become tedious very quickly.

Note: Do not panic! The default kernels that are installed with most distributions come with a selection of drivers that cover all the common network cards. You probably won't have to do more than load and unload a few modules until you get it right.

There's not enough space here for a lesson in kernel/module compilation, so I will assume that you have a system with networking enabled in the kernel (as it is in all the default kernels shipped with any distribution I ever heard of) and with a selection of modules, which hopefully includes the one to match your card.

IP addresses and netmasks

An IPv4 address is a 32 bit number, which is usually shown in dotted quad notation: four decimal numbers separated by full stops (e.g. 192.168.10.5). This number is further split into two parts, one identifying the network on which the host (machine) is located and one uniquely identifying the host within that network.

You can't tell which part is which just by looking at the address (some IP addresses are commonly split in particular ways but don't rely on this). To identify the network and host parts of an IP address, you need to know the netmask for the network on which the IP address is located. The netmask specifies which bits of an address are the network address and which are the host address. The 32 bits may be divided in any permutation but it is most convenient and least confusing simply to split them into two blocks, with the high block of bits representing the network and the low block the host. Network bits have a value of 1 and host bits a value of 0, so that AND-ing an IP address and its netmask give you the network address.

In our example network, the highest 24 bits of the address are the network
address. So the netmask could be represented by the binary number 11111111111111111111111100000000. This is obviously not convenient, so netmasks are usually shown in one of two ways:

- In dotted quad notation. Our example netmask would be 255.255.255.0
- A slash and a decimal digit, appended to an IP address, the digit showing how many high bits comprise the network address. So 192.168.10.0/24 would indicate that the first three bytes are the network address and that the final, least significant byte is used to allocate host addresses. Since one address is reserved as a broadcast address, that leaves 255 host addresses we can use.

So why all this trouble? The answer is that before an IP packet can be delivered to its destination we need to know if it is on the local network or a remote one.

The network address may be further divided into a network number and a subnet number. This distinction will be explored in the next article: for the purposes of this article no distinction is made.

Select a module

If you didn't identify the correct driver when doing your Google search, now is the time. Compare the information you found then with the help in */usr/src/linux/documentation/networking*. If you find more than one match, don't panic, just repeat the process described below until you find one that works (you can leave fancy comparison-testing till you know more). Be sure to note any special parameters that may need to be passed to the module (another good reason not to buy cheap: decent modern cards usually autoconfigure).

Load the module

Say you want to try the eepro100 module. Load it:

```
# modprobe eepro100
```

Now, check the end of */var/log/messages* for any messages from the module. If the module didn't work then you will see obvious error messages: try another module or try reloading this one with different parameters. A successful load will give output like this:

```
Apr 1 20:15:33 localhost kernel: eth0: ➥
Intel EtherExpress Pro 10/100
Apr 1 20:15:33 localhost kernel: Board ➥
assembly 689661-004
Apr 1 20:15:33 localhost kernel: General ➥
self-test: passed
Apr 1 20:15:33 localhost kernel: Serial ➥
sub-system self-test: passed
Apr 1 20:15:33 localhost kernel: Internal ➥
registers self-test: passed
Apr 1 20:15:33 localhost kernel: ROM ➥
checksum self-test: passed
Apr 1 20:15:33 localhost kernel: Receiver ➥
lock-up workaround activated.
```

Success! Light a cigar (away from the equipment). But note that eth0 in the first line, we need it for the next step. What does it mean? It's the interface name assigned to the card.

Boo-boos

If you compiled the driver into the kernel, smack yourself on the wrist and prepare for multiple reboots. Each time you reboot, check the messages that scroll past. Too fast? Once it's finished booting, log in and check the contents of */var/log/dmesg*. You should find the output from the driver somewhere in there.

Configuring the network interface

Loading the correct driver gets us halfway there. Now we need to configure and activate the network

Debian network config file

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)

# The loopback interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
   address 192.168.10.10
   netmask 255.255.255.0
```

connection. For this example, I am going to assume we are connecting to a network with address 192.168.10.0/24 and that this computer will have the address 192.168.10.10 on that network. If this means nothing to you then you should read the boxout on IP addresses and netmasks.

To do this, we use the *ifconfig* command, giving as parameters the interface name, the address and netmask:

```
# ifconfig eth0 192.168.10.10 netmask ➥
255.255.255.0
```

If you get no feedback then in good *nix fashion this means that nothing went wrong. So now you should list the interfaces on your computer, by running the *ifconfig* command with no parameters. The output should include a record like this:

```
eth0    Link encap:Ethernet  HWaddr ➥
00:01:02:87:18:AB
   inet addr:192.168.10.10  ➥
Bcast:192.168.10.255  Mask:255.255.255.0
   UP BROADCAST RUNNING MULTICAST  MTU:1500  ➥
Metric:1
   RX packets:819780 errors:116 dropped:0 ➥
overruns:0 frame:232
   TX packets:593233 errors:0 dropped:0 ➥
overruns:0 carrier:69
   collisions:230 txqueuelen:100
   RX bytes:873172906 (832.7 MiB)  TX ➥
bytes:48467017 (46.2 MiB)
   Interrupt:9 Base address:0xec00
```

Before you light yourself another cigar, let's try pinging the address of another network host (the ping command sends a stream of IP packets to the requested address and reports their progress). On this example network, there's a router at 192.168.10.1, so:

```
# ping 192.168.10.1
PING 192.168.10.1 (192.168.10.1): 56 data bytes
64 bytes from 192.168.10.1: icmp_seq=0 ➥
ttl=255 time=0.2 ms
64 bytes from 192.168.10.1: icmp_seq=1 ➥
ttl=255 time=0.2 ms
64 bytes from 192.168.10.1: icmp_seq=2 ➥
ttl=255 time=0.2 ms
64 bytes from 192.168.10.1: icmp_seq=3 ➥
ttl=255 time=0.2 ms
64 bytes from 192.168.10.1: icmp_seq=4 ➥
ttl=255 time=0.2 ms
64 bytes from 192.168.10.1: icmp_seq=5 ➥
ttl=255 time=0.2 ms
64 bytes from 192.168.10.1: icmp_seq=6 ➥
ttl=255 time=0.2 ms
```

Success! You managed to connect to a network and talk to another host on the network – the aim of this introductory article. But we're not finished yet.

Red Hat interface config script

```
# /etc/sysconfig/network-scIpts/ifcfg-eth0

DEVICE=eth0
USERCTL=no
ONBOOT=yes
BOOTPROTO=
BROADCAST=192.168.10.255
NETWORK=192.168.10.0
NETMASK=255.255.255.0
IPADDR=192.168.10.10
```

Up and down

OK, now that you have configured and activated your interface, you can deactivate it any time you like. To bring it down run:

```
# ifconfig eth0 down
```

Now run *ifconfig* with no arguments – you'll see that the eth0 record has disappeared. You can bring the interface up again by running the above command but replacing *down* with *up*. Try it and then run *ifconfig* again to see if the interface has come back up with the same settings.

Achieving permanence

What we have achieved so far is fair enough, but it won't survive a reboot and even Linux boxes have to be shutdown once in a while (only for hardware maintenance, obviously).

So what do you do if you want this interface to be configured automatically when the machine restarts? You could create a script to configure the interface and add it to the *sysvinit* scripts. But you don't have to: while there is no definitive Linux standard for this, there is a semi-standard. Most distributions enable you to record the interface's details in a text config file, to bring the interfaces so defined up and down with the *ifup* and *ifdown* tools and to specify which interfaces should be activated on startup. Example configuration files for Debian and Red Hat are shown in the two boxouts, Debian network config file and Red Hat interface config script. In both cases, the interface we created above would be configured and activated as part of the startup process.

Summary

This article has shown you how to get a simple network connection up and running. I hope it has been enough to get you interested and experimenting. But there's still a lot we haven't covered. The next article will explain subnets and routes, show the difference between hubs, switches and routers, introduce the IP tool (brought in with the 2.2.x series of kernels) and explain how you can turn a Linux box into a network router/gateway. See you then.