The Essentials of Linux Network Administration

In this Paper

Introduction

If you're ever going to do anything interesting with Linux, just like any other OS, you need to be connected to a network, whether it's your own local company network or the public Internet. In this paper to learn what you need to know to connect your Linux host to the network as well as some tools to help you troubleshoot if things don't go exactly as expected.

Here's what you'll learn:

- The different types of network interfaces in Linux
- How to configure IP addressing
- Networking troubleshooting tools available in Linux
- How to connect multiple network interfaces together to form a bond

We'll kick it off by talking about Linux network interfaces.

Understanding Linux Network Interfaces

Different versions of Linux may name network interfaces differently (see the callout about how Linux network device names are changing). In general, just about all Linux operating systems will have at least two network interfaces. They are:

- **Loopback.** The *loopback* (lo) interface will have an IP address of 127.0.0.1, which represents the host itself. Suppose you want to open a web page running on the same Linux server you are on. You could open http://127.0.0.1 in your web browser. That IP address won't be accessible over the network.
- **Ethernet.** The *ethernet 0* (eth0) interface is typically the connection to the local network. Even if you are running Linux in a virtual machine (VM), you'll still have an eth0 interface that connects to the physical network interface of the host. Most commonly, you should ensure that eth0 is in an UP state and has an IP address so that you can communicate with the local network and likely over the Internet.

Predictable network interface naming convention

If you've been in IT for any length of time, you know that the only constant is change, and that applies to network interface naming conventions as well. Up until very recently, the only naming convention you had to worry about was the one that was just presented. However, with systems featuring systemd v197 or later, common network names such as "eth0" will be assigned more predictable names based on what is known as the *predictable network interface* naming convention. So, rather than an interface named eth0, you may have one named ens3 or enp0s3. Ironically, under older versions of systemd, there was predictability about the existence of eth0. Under the Predictable Network Interface naming guidelines, however, there isn't a guaranteed standard network interface name across systems. The new scheme does have a number of benefits, the most important of which is that the physical interface to interface name association will survive reboots, even if you add hardware. Under the old system, if you added a network adapter, you ran the risk of automatically changing interface names, which had serious consequences for your configuration. For the purposes of this paper, we're sticking with the legacy naming convention, but as you continue your Linux networking journey, the predictable network interface naming convention is the new paradigm you need to be aware of.

For more information this change, please visit https://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/

The Linux command to configure network interfaces/ devices/ links (whatever term you use) is **`ip link`**. In the following example, you can see how **`ip link`** (with no other options) shows two different interfaces, their status, and their MAC addresses associated with each one (we'll talk more about MAC addresses, next page):

```
david@debian:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT
group default qlen 1000
    link/ether 00:50:56:a3:71:f5 brd ff:ff:ff:ff:ff:ff
```

The **ip link** command is also used to configure network interfaces. For example, you can change the status of interfaces with **ip link set [dev] { up | down }**. You can also reconfigure network interfaces with a command like **ip link set lo mtu 1500**.

For more information on the **ip link** command use **man ip link**.

MAC Addresses

A *media access control* (MAC) address is the unique identifier assigned to a network interface at layer 2—the Data Link layer—of the OSI Model. A network interface always has a MAC address—often referred to as the *hardware address*—even if it does not have an IP address. MAC addresses are assigned at the time that a network adapter is manufactured or, if it's a virtualized network adapter, the time that the adapter is created and appears as six groups of two hexadecimal digits each. On the Ethernet interface, eth0, shown above, the MAC address is also called the *link* or *ether address*. In the **ip link** output above, you can see that the MAC address in this case is *00:50:56:a3:71:f5*.

But what if you want to know the IP addresses of these network interfaces? That's next!

Going back to the old

If you *really* want or need to use a deprecated command such as **ifconfig, arp,** or **route**, you still can. You just have to install the net-tools package on your system. On a Debian system, like the one shown in these examples, as a privileged (root) user, run **apt-get install net-tools**. If you're using a Linux distribution that uses yum as a package manager, such as CentOS, you can install net-tools using the **yum -y install net-tools** command.

If you're only doing this because you're comfortable with the old ways, however, we recommend that you start to phase out your use of these old commands because there's no guarantee that they'll be around forever, they aren't kept up to date, and they may not support all the features of the new commands.

IP Addressing

They are unique on the same network, every device has at least one, and addresses typically fall somewhere between 1.1.1.1 and 255.255.255.255. What are they? IP addresses, of course! Let's assume that you already know the basics around TCP/IP, and we'll focus on how to work with them in Linux. Later, we'll talk about how to configure IP addresses on your Linux machine.

To view IP addresses, use the **ip address** command, or just **ip addr**, like this:

```
david@debian:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default
qlen 1000
    link/ether 00:50:56:a3:71:f5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.80/24 brd 192.168.1.255 scope global dynamic eth0
```

As you can see, the IP address of the loopback interface is 127.0.0.1. What's the address of the eth0 interface in this case? The eth0 "inet" address (IPV4 address) is 192.168.1.80, and it's a dynamic address, received via DHCP, which we'll talk about below.

When it comes to Linux networking tools, there is one that just about everyone has heard of, and that is **ping.** Ping, which began life as an acronym but now enjoys its status as a full-fledged word, is the most basic network test tool around for testing network reachability. It sends out an *Internet Control Message Protocol* (ICMP) packet across the network and notifies you whether there is a response. If a host is up and able to communicate on the network, an ICMP response will be returned. If, however, a host is not reachable, you will get a notice that the host was unreachable or timed out (meaning that the ping test failed). Here's an example of a host that is unreachable:

```
david@debian:~$ ping -c5 192.168.192.196
PING 192.168.192.196 (192.168.192.196) 56(84) bytes of data.
--- 192.168.192.196 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4018ms
```

(The "-c5" was used to send just five ping packets; otherwise, ping will continue forever.)

In these results, five packets were transmitted, and all of them received no response, so there was 100% packet loss. What that means is that this host is unreachable, or down.

Another common Linux network troubleshooting tool is **traceroute**. Traceroute probes the network between the local system and a destination, gathering information about each IP router in the path. The traceroute command is useful when you think there may be a network issue, such as a host down along a path or a slow response from one of the intermediary nodes, and you want to find out which node is creating the problem.

Here's an example:

```
david@debian:~$ traceroute www.apple.com
traceroute to www.apple.com (23.46.180.139), 30 hops max, 60 byte packets
 1  192.168.1.1 (192.168.1.1)  0.225 ms  0.273 ms  0.283 ms
 2  10.10.0.1 (10.10.0.1)  11.046 ms  11.938 ms  16.645 ms
 3  pool.hargray.net (64.202.123.123)  28.169 ms  22.060 ms  21.785 ms
 4  10ge14-8.core1.atl1.he.net (216.66.49.77)  22.552 ms  22.391 ms  19.566 ms
 5  atx-brdr-01.inet.qwest.net (63.146.26.69)  23.189 ms  21.705 ms  21.952 ms
 6  a23-46-180-139.deploy.static.akamaitechnologies.com (23.46.180.139)  21.116 ms
21.365 ms  19.497 ms
```

But why are some of those addresses listed on the left actually names instead of IP addresses? That's because *domain name system* (DNS) is replacing the IP with a friendly name. You'll learn about DNS in later in this paper.

Remember, every Linux command shown here has verbose instructions on how to use it in the man page. Just type **man *commandname*** to learn more.

DHCP

What if you have dozens, hundreds, or thousands of computers on your network? It would be incredibly time-consuming to manually assign IP addresses and to actually track which machines have which IP address. That's where the *dynamic host configuration protocol* (DHCP) comes in. DHCP is used to obtain an IP address when a host or device first comes on the network.

DHCP is commonly used for client systems or devices that don't experience any side effects from a periodically changing IP address. On server systems, administrators either manually configure static IP addresses, or they create what are known as static DHCP reservations that are tied to the MAC address of the network adapter. These static reservations ensure that the network adapter will get the same IP address every time it restarts.

Here's how the typical DHCP process works:

1. When a computer starts up, it sends a DHCP request out on the network.
2. Assuming a DHCP server is present, a DHCP server responds with the IP address configuration for that device.
3. That IP address is marked as reserved so that it's not accidentally assigned to some other device.

To learn more about the exact packets that make up the process of obtaining an IP address, see this diagram: http://www.smartpctricks.com/wp-content/uploads/2014/04/DHCP-Packets-Establishment.png.

Note that the prior text said, "IP address configuration" and not just "IP address." The IP configuration that is returned to a requesting client contains, at a minimum, the IP address, the IP subnet mask, the IP default gateway, and DNS server details. Most end user devices are configured to use DHCP.

Most operating systems, including Linux, are configured to use a DHCP client to obtain their initial IP address. You can tell an interface is using DHCP if its IP address is set to **DYNAMIC,** as in the output below.

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default
qlen 1000
    link/ether 00:50:56:a3:71:f5 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.80/24 brd 192.168.1.255 scope global dynamic eth0
```

The local configuration file for the DHCP client (called **dhclient**) is at **/etc/dhcp/dhclient.conf.** This is a configuration file that dictates to Linux how it will receive IP configuration information from a DHCP server. To check the status on the DHCP client, you can **cat** the syslog (system log file) and **grep** for dhcp, like this:

```
david@debian:~$  sudo grep -Ei dhcp /var/log/syslog
May 15 08:37:44 debian dhclient: DHCPREQUEST on eth0 to 192.168.1.1 port 67
May 15 08:37:44 debian dhclient: DHCPACK from 192.168.1.1
May 15 08:37:44 debian NetworkManager[425]: <info> (eth0): DHCPv4 state changed
renew -> renew
May 15 08:37:44 debian nm-dispatcher: Dispatching action 'dhcp4-change' for eth0
```

You can find more details on the DHCP client leases in the files /var/lib/dhcp/*.leases

DNS

Computers that connect to each other using TCP/IP (the most prevalent form of connection protocol) talk with each other using IP addresses; however, it would be really painful to have to remember the IP address of everything you want to connect to. Imagine having to recall the IP address of Google each time you wanted to search the web. *Domain name system* (DNS) is used to map IP addresses to names. Everyone is familiar with using their web browser, entering a friendly name like google.com or apple.com, and being taken to the company's website without ever having to type an IP address. It's DNS behind the scenes that is mapping that friendly name to an IP address by doing a DNS lookup. To find out if your Linux host is using DNS, we will be running through some troubleshooting commands, such as **dig** and **nslookup**, later.

That being said, the basics of DNS in Linux are this:

- A local file called /etc/hosts is used for the first point of lookup for any host name prior to going out to a DNS server on the network. If the name is found there, no further searches are performed. As the superuser, you have the option to edit the hosts file and configure a static name to IP address mapping.
- The /etc/resolv.conf file shows the local domains to be searched and what server names to use for DNS resolution.

For example, here's what a sample **resolv.conf** file looks like:

```
david@debian:~$ sudo cat /etc/resolv.conf
search wiredbraincoffee.com wiredbraincoffee.com.
nameserver 192.168.1.1
```

In this case, the default domain is wiredbraincoffee.com, and the only name server is 192.168.1.1.

DNS Resolution

By default, DNS name resolution works as described here, but is very modular. The hosts portion of /etc/nsswitch.conf can include directory services like NIS+ or LDAP as well.

When it comes to troubleshooting DNS, you should be aware of the following important tools:

- **`dig`.** The *domain Internet groper*, or dig, performs verbose DNS lookups and is great for troubleshooting DNS issues.
- **`getent ahosts`**. The getent tool with the ahosts option enumerates name service switch files, specifically for host entries.
- **`nslookup`.** The name server lookup, or nslookup, performs a variety of different DNS server lookups: mail server lookups, reverse lookups, and more. It's commonly used to look up the IP address of a host.

Network Statistics and Counters

When performing network troubleshooting, it's always good to gather some statistics to answer questions like, "Is the network interface even transmitting any data? Is the interface taking errors? What process is sending all that traffic?"

Here's an example of the **`netstat`** command displaying active processes that have active network interface connections:

Here's an example of the **`ip -s link`** command showing us the statistics for our network links:

```
david@debian:~$ ip -s link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default
  link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
  RX: bytes  packets  errors  dropped overrun mcast
  63339      505      0       0       0       0
  TX: bytes  packets  errors  dropped carrier collsns
  63339      505      0       0       0       0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP mode DEFAULT
group default qlen 1000
  link/ether 00:50:56:a3:71:f5 brd ff:ff:ff:ff:ff:ff
  RX: bytes  packets  errors  dropped overrun mcast
  410864536  1342597  0       0       0       925004
```

```
TX: bytes  packets  errors  dropped carrier collsns
20398071   163673   0       0       0       0
```

Here's an example of the **`netstat`** command showing us what our active processes are that have the network interface open:

```
david@debian:~$ netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address  Foreign Address  State
tcp   0      0      debian:ssh     iMac:52985       ESTABLISHED
tcp   0      0      debian:40980   192.168.1.128:37518 TIME_WAIT
tcp6  1      0      localhost:33904 localhost:ipp   CLOSE_WAIT
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags  Type   I-Node   Path
unix  20     [ ]    DGRAM  8963     /run/systemd/journal/dev-log
unix  6      [ ]    DGRAM  8972     /run/systemd/journal/socket
unix  2      [ ]    DGRAM  15451    /run/user/1000/systemd/notify
```

(**output truncated**)

And here's an example of the **`netstat -l`** command that shows us the active listening services on this host:

```
david@debian:~$ netstat -l
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address      Foreign Address    State
tcp        0      0 localhost:mysql    *:*                LISTEN
tcp        0      0 *:48875            *:*                LISTEN
tcp        0      0 *:sunrpc           *:*                LISTEN
tcp        0      0 *:ssh              *:*                LISTEN
tcp        0      0 localhost:ipp      *:*                LISTEN
Active UNIX domain sockets (only servers)
Proto RefCnt Flags   Type      State     I-Node Path
unix  2      [ ACC ] SEQPACKET LISTENING 8197   /run/udev/control
unix  2      [ ACC ] STREAM    LISTENING 8200   /run/systemd/journal/stdout
unix  2      [ ACC ] STREAM    LISTENING 15895  /run/user/1000/keyring/pkcs11
```

(**output truncated**)

So far, we have covered a lot about how to view and show network information, but nothing about how to change network configurations. Let's cover some of the most common network configurations that someone new to Linux might want to perform.

We'll start off with changing an IP address. When it comes to making any changes in Linux, keep in mind that you can make two types of changes:

- Changes that are immediately effective but are *non-persistent* (meaning they won't survive a restart of the operating system)
- Changes that are effective after the next restart of the OS, known as *persistent changes*

To make an immediately effective change in your IP configuration, you use the **ip** command with its variety of command options such as **link, route,** and **address**. To use the **ip** command set, you'll have to have the iproute2 package installed. It usually is installed by default, but that depends on which version of Linux you are using.

Here we use the **ip address** command, like this:

```
root@debian:/home/david# ip address add 10.10.10.10/8 dev eth0
```

However, once the Linux machine is restarted, the default IP address will be back on interface eth0.

To make this IP address change persistent on a Debian or Ubuntu system, you need to edit the file /etc/network/interfaces and add the configuration for eth0. To edit this file, use the nano command like this: nano /etc/network/interfaces. If you are using CentOS or RHEL (Red Hat Enterprise Linux), the same configuration is found in the /etc/sysconfig/network-scripts directory.

To make the IP address change take effect, you can either reboot the host or use the **ifdown/ifup** commands. At that point, the **ip address** command output might look like this:

```
david@debian:~$ ip address show dev eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN group
default qlen 1000
    link/ether 00:0c:29:d0:e8:7e brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.72/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20c:29ff:fed0:e87e/64 scope link
       valid_lft forever preferred_lft forever
```

You can see that the new IP address has been added, and making the change this way ensures that it is persistent after the Linux OS restart.

Adventures in editing text files

Editing of text files in Linux is usually done with the **vi**, **emacs**, or **nano** commands. The "battle" between vi and emacs is long running, and their followers are fanatical. The vi text editor has been around since 1976 and is known for its lack of user-friendliness. With vi you must already know, or be willing to read the documentation on, how to use specific command options to even start editing text. Similar to vi, emacs has also been around since the 1970's and isn't known for its ease of use. Contrary to vi, emacs has so many plugins (including terminal, calculator, email, and web browser) that it feels like its own operating system. Nano, on the other hand, was created in 1999 and is very user-friendly because it works just like a traditional word processing application. For those who are new to editing text in Linux, my recommendation is to use nano.

To recap, here are some of the common network changes that you will probably want to make in order for your Linux networking configuration to persist across reboots:

- Change the state of network interfaces by using the ifdown / ifup command set.
- Add DNS name servers to an interface by editing the /etc/resolv.conf file or the /etc/network/interfaces file.
- Change the hostname of the server by editing the /etc/hostname and /etc/hosts file, and verifying it with **`hostname -f`**.

Network Interface Bonding

There are times when you need more bandwidth than a single interface can provide, or you want some form of link redundancy in case of a cabling or other network problem. This link redundancy function goes by many names, depending on the vendor: EtherChannel, VMware PortGroups, Bonds, and Link Aggregation Groups (LAG) are just a few. Linux also provides this capability and calls it *bonding*. It allows you to create a single logical network link that is comprised of multiple physical links and that scales up as you add more interfaces, can provide load balancing across the interfaces, and can provide failover protection.

ifdown and ifup commands

The **`ifdown`** and **`ifup`** commands are used to restart an interface without having to restart a whole server. Once you make a change to a network interface configuration, you need to force that interface to reread its configuration file. You accomplish this by bringing the interface down with the ifdown command and then bringing it back up with the ifup command. When the ifup command is executed, the configuration file is reread and the interface is brought back into operation with its newly minted parameters.

By the way, the ifdown and ifup commands aren't included in the default path on all Linux distributions. As such, you may have to explicitly include the full path to the command. For example, to bring down the eth0 interface, you may need to type /sbin/ifdown eth0. Likewise, to bring the interface back up, you may need to type /sbin/ifup eth0.

The /etc/network/interfaces file is used to tell ifup and ifdown how to configure network interfaces as those interfaces are brought up and down. For example, you would configure an interface with a static IP address by entering the details in the interfaces file. More information on the interfaces file can be found by typing **`man interfaces`**.

A word of warning: If you're connected to your Linux server via SSH, be mindful of which interface you're working with. If you accidentally bring down the network interface that your SSH connection is using, you might lose remote access to the server.

To use network bonding, you need to install the bonding kernel module via the **modprobe** command. Modprobe allows you to add additional capability to the Linux kernel. It works like this:

```
david@debian:~$ sudo modprobe bonding
```

At this point, you can create a bond using the iproute2 tools, which allow you to establish the bond as well as set its mode. You can get some hints with **ip link help** and **ip link help bond**.

Please note that you may read about **ifenslave** when searching the Internet; however, that tool has been deprecated with the iproute2 tools taking its place. You will find that many once-common Linux commands become obsolete over time, so make sure to stay current!

You can put interfaces eth0, eth1, and eth2 into a bond like this (Figure 1):

```
david@debian:~$ sudo ip link add bond0 type bond mode 802.3ad
david@debian:~$ sudo ip link set eth0 master bond0
david@debian:~$ sudo ip link set eth1 master bond0
david@debian:~$ sudo ip link set eth2 master bond0
```

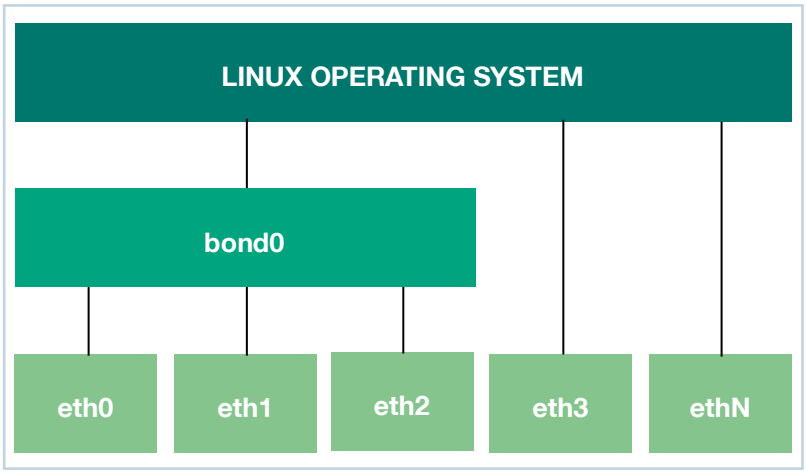

Figure 1. Multiple Ethernet interfaces bonded into a single network interface

One of the most common parameters to set when creating a bond is the "mode," which is how the bond interacts with the connected network.

Bonding, when incorrectly configured or cabled, can be the source of some pretty messy network problems; 802.3ad mode runs the Link Aggregation Control Protocol (LACP) with the switch or server at the other end of the bond to make sure that everything is connected correctly.

Linux network bonding modes

Linux currently supports the following bond modes:

- **balance-rr**. The default, round-robin bonding, which provides load balancing and fault tolerance
- **active-backup**. Provides fault tolerance whereby only one slave can be active at a time and, if it fails, the other slave takes over
- **balance-xor**. Provides fault tolerance and load balancing by transmitting based on hash
- **broadcast**. Provides fault tolerance by transmitting everything on all slave interfaces
- **802.3ad**. IEEE 802.3ad standard for dynamic link aggregation creates aggregation groups for links that share the same speed and duplex in order to provide for fault tolerance and load balancing. 802.3ad is one of the most common types of bonding. 802.3ad uses LACP to communicate with the other side of the bond.
- **balance-tlb**. Adaptive transmit load balancing that doesn't require any special switch support
- **balance-alb**. Adaptive load balancing that doesn't require any special switch support due to its use of ARP negotiation

The most common modes are active-backup and 802.3ad.

In Summary

You just learned about network interfaces, DHCP, DNS, IP address configuration, interface bonding, and more. Whether you need to configure your figure Linux host or you need to troubleshoot a Linux networking issue on the first day of your new job, you won't be able to accomplish either until you've learned how to identify network interfaces, check their status, and reset them when needed. After reading this paper, you should know the essentials of Linux network administration.

Last updated: December 2017