

Configuring TCP/IP under Linux

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

1. About this tutorial	2
2. Understanding the TCP/IP protocol	3
3. IP addressing, subnets, and routing	6
4. TCP/IP configuration files (Red Hat 7.0)	11
5. Configuring network interfaces and routes	16
6. Monitoring your TCP/IP network	19
7. Summary, resources, and feedback	22

Section 1. About this tutorial

Should I take this tutorial?

This tutorial reviews the origins of TCP/IP, then moves into the model around which it was developed, and finally covers how TCP/IP works -- including IP addresses, subnets, and routing. With this theoretical foundation laid, we discuss the various network configuration files required by Linux, how to initialize a network interface, and how to edit the system's routing table. The tutorial closes with a brief look at how to analyze your network and ensure that data gets to where it's supposed to go, without error.

About the author

Tom Syroid is a technology adventurer who is currently working as a staff writer for a leading author's agency. He co-authored *Outlook 2000 in a Nutshell* (O'Reilly) and *OpenLinux Secrets* (IDG), and specializes in *NIX system administration, system security, Apache deployment and configuration, and file sharing across diverse platforms.

Tom can be reached for query or comment at tom@syroidmanor.com.

Section 2. Understanding the TCP/IP protocol

A brief history of TCP/IP

The suite of widely used protocols known as Transmission Control Protocol/Internet Protocol (TCP/IP) has become the de facto standard for network communications in recent years. This is due in large part to the explosive growth of the Internet, and the need for diverse platforms, devices, and operating systems to share data in a "language" everyone understands. Let's start with a look at TCP/IP's history.

In the late 1960s, the U.S. Department of Defense (DOD) recognized a communication problem developing within its halls. Traffic from the ever-increasing volume of electronic information among DOD staff, research labs, universities, and contractors had hit a major obstacle. The various entities and organizations comprising DOD had computer systems from different computer manufacturers, running different operating systems, using different networking topologies and protocols.

The Advanced Research Projects Agency (ARPA) was assigned the task of coming up with a solution to the problem. ARPA formed an alliance with universities and computer manufactures to develop a set of communication standards. This alliance specified and built a four-node network that became the foundation of today's Internet. During the 1970s, this network migrated to a new, core protocol design that became the basis for TCP/IP.

The Open System Interconnection (OSI) model

Many different types of computers are in use today, varying in operating systems, CPUs, network interfaces, etc. These differences make communication between computer systems problematic. In 1977, the International Organization for Standardization (ISO) created a subcommittee to develop data communication standards to promote multi-vendor interoperability. The result was the Open System Interconnection (OSI) model.

The OSI model doesn't specify any communication standards or protocols; instead, it provides guidelines that communication tasks follow.

NOTE: It's important to understand that the OSI model is simply a model, or a framework, that specifies the functions to be performed. It doesn't detail how these functions are performed. ISO, however, does certify specific protocols that meet OSI standards for parts of the OSI model. For example, the CCIT X.25 protocol is accepted by ISO as an implementation that provides most of the services of the Network layer of the OSI model.

The seven OSI layers

To simplify matters, the ISO subcommittees took the divide-and-conquer approach. By dividing the complex communication process into smaller subtasks, the problem became more manageable, and each subtask could then be optimized individually. The OSI model is composed of seven layers:

- * Application
- * Presentation
- * Session
- * Transport
- * Network
- * Data Link
- * Physical

Each layer is assigned a specific set of functions. Each layer uses the services of the layer beneath it and provides services to the layer above it. For example, the Network layer uses services from the Data Link layer and provides network-related services to the Transport layer.

OSI layers, continued

The concept of a layer making use of services and providing services to its adjacent layers is simple. Consider how a company operates: the secretary provides secretarial services to the president (the next layer up) to write a memo. The secretary uses the services of a messenger (the next layer down) to deliver the message. By separating these services, the secretary (application) doesn't have to know how the message is actually carried to its recipient. The secretary merely has to ask the messenger (network) to deliver it. Just as many secretaries can send memos in this way by using a standard messenger service, a layered network can send packets by handing them to the network layer for delivery.

NOTE: Don't confuse the Application layer with application programs you execute on a computer. Remember that the Application layer is part of the OSI model that doesn't specify how the interface between a user and the communication pathway happens; an application program is a specific implementation of this interface. A real application typically performs Application, Session, and Presentation layer services and leaves Transport, Network, Data Link, and Physical layer services to the network operating system.

Communicating across layers

Each layer communicates with its peer in other computers. For example, layer 3 in one system communicates with layer 3 in another computer system.

When information is passed from one layer down to the next, a header is added to the data to indicate where the information is coming from and going to. The header-plus-data block of information from one layer becomes the data for the next. For example when layer 4 passes data to layer 3, it adds its own header. When layer 3 passes the information to layer 2, it considers the header-plus-data from layer 4 as data and adds its own header before passing that combination down.

In each layer, the information units are given different names:

Application --> Message

Transport --> Segment

Network --> Datagram

Data Link --> Frame (also called packet)

Physical --> Bit

Before the advent of the OSI model, the U.S. Department of Defense defined its own networking model, known as the DOD model. The DOD model is closely related to the TCP/IP suite of protocols, as explained in the next section.

The TCP/IP protocol stack (TCP)

The TCP/IP protocol stack represents a network architecture that's similar to the OSI model.

TCP/IP does not, however, make detailed distinctions between the top layers of the protocol stack as the OSI model does. The top three OSI layers are roughly equivalent to the Internet process protocols. Some examples of process protocols are Telnet, FTP, SMTP, NFS, SNMP, and DNS.

The Transport layer of the OSI model is responsible for reliable data delivery. In the Internet protocol stack, this corresponds to the host-to-host protocols. Examples of these are TCP and UDP. TCP is used to translate variable-length messages from upper-layer protocols and provides the necessary acknowledgment and connection-oriented flow control between remote systems.

UDP is similar to TCP, except that it's not connection oriented and doesn't acknowledge data receipt. UDP only receives messages and passes them along to the upper-level protocols. Because UDP doesn't have any of the overhead related to TCP, it provides a much more efficient interface for such actions as remote disk services.

The TCP/IP protocol stack (IP)

The Internet Protocol (IP) is responsible for connectionless communications between systems. It maps onto the OSI model as part of the Network layer, which is responsible for moving information around the network. This communication is accomplished by examining the Network layer address, which determines the systems and the path to send the message.

IP provides the same functionality as the Network layer and helps get the messages between systems, but it doesn't guarantee the delivery of these messages. IP may also fragment the messages into chunks and then reassemble them at the destination. In addition, each fragment may take a different network path between systems. If the fragments arrive out of order, IP reassembles the packets into the correct sequence at the destination.

Section 3. IP addressing, subnets, and routing

IP addresses

The Internet Protocol requires an address be assigned to every device on a network. This address, known as the IP address, is organized as a series of four octets. These octets each define a unique address, with part of the address representing a network (and optionally a subnetwork) and another part representing a particular node on the network.

Several addresses have special meanings on TCP/IP networks:

- * An address starting with a zero references the local node within its current network. For example, 0.0.0.23 references workstation 23 on the current network. Address 0.0.0.0 references the current workstation.
- * The addresses starting with 127 are important in troubleshooting and network diagnosis. The network address block 127.x.x.x is formally defined as class A addresses, which are reserved for internal loopback functions.
- * The ALL address is represented by turning on all bits, giving a value of 255. Therefore, 192.18.255.255 sends a message to all nodes on network 192.18.; similarly, 255.255.255.255 sends a message to every node on the Internet. These addresses are used for multicast messages and service announcements.

CAUTION: When you assign node numbers to a system, don't use 0 or 255; these are reserved numbers and have special meaning.

IP address classes

IP addresses are assigned in ranges referred to as classes, depending on the application and the size of an organization. The three most common classes are A, B, and C. These three classes represent the number of locally assignable bits available for the local network.

Class A addresses are used for very large networks or collections of related networks. Class B addresses are used for large networks having more than 256 nodes (but fewer than 65,536 nodes). Class C addresses are used by most organizations. It's a better idea for an organization to get several class C addresses because the number of class B addresses is limited. Class D is reserved for multicast messages on the network, and class E is reserved for experimentation and development.

Class Addresses

A 0.x.x.x to 126.x.x.x
B 128.0.x.x to 191.255.x.x
C 192.0.0.x to 223.255.255.x
D 224.0.0.1 to 239.255.255.255
E 240.x.x.x to 255.255.255.255

"Private" IP addresses

If your network is not connected to the Internet and won't be in the near future, you are free to choose any legal network address. Just make sure no packets from your internal network escape to the real Internet. To make sure no harm could be done even if packets did escape, you should use one of the network numbers reserved for private use. The Internet Assigned Numbers Authority (IANA) has set aside several network numbers from classes A, B, and C that you can use without registering. These addresses are valid only within your private network and are (theoretically) not routed between real Internet sites.

A 10.0.0.0
B 172.16.0.0 - 172.31.0.0
C 192.168.0.0 - 192.168.255.0

Note that the second and third blocks contain 16 and 256 networks, respectively. Picking your addresses from one of these network numbers is useful for networks completely unconnected to the Internet. And you can still implement access to another network (such as the Internet) by using a single host as a gateway. To your local network, the gateway is accessible by its internal private IP address, while the outside world knows it by an officially registered address (assigned to you by your bandwidth provider).

Subnetworks and subnet masks

Subnetting is the process of dividing a large real network into smaller logical networks. Reasons for dividing a network include the electrical (physical layer) limitations of the networking technology, a desire to segment for simplicity by putting a separate network on each floor of a building (or in each department or for each application), reducing network segment loads, or a need for remote locations connected with a high-speed line.

The resulting networks are smaller chunks of the whole and are easier to manage. Smaller subnets communicate among one another through gateways and routers. Also, an organization may have several subnetworks that are physically on the same network, so as to logically divide network functions into workgroups.

The individual subnets are a division of the whole. Suppose that a class B network is divided into 64 separate subnets. To accomplish this subnetting, the IP address is viewed in two parts: network and host. The network part becomes the assigned IP address and the subnet information bits. These bits are, in essence, removed from the host's part of the address. The assigned number of bits for a class B network is 16. The subnet part adds 6 bits, for a total of 22 bits to distinguish the subnetwork. The division results in 64 networks with 1,024 nodes in each. The network part can be larger or smaller, depending on the number of networks desired or the number of nodes per network.

Subnets, part 2

Setting a subnet mask is a matter of determining where the network address ends and the host address begins. The subnet mask contains all 1s in the network field and 0s (zeroes) in the host field.

Suppose a class C network is composed of the following:
N = network

H = host

NNNNNNNN.NNNNNNNN.NNNNNNNN.HHHHHHHH

Each position represents a single bit out of the 32-bit address space. If this class C network is to be divided into four class C networks, the pattern resembles the following:

NNNNNNNN.NNNNNNNN.NNNNNNNN.NNHHHHHH

The subnet mask looks like the following:

11111111.11111111.11111111.11000000

If this address is written in base-10 dot notation (a.k.a. *dotted quad*), the subnet mask is 255.255.255.192. This mask is used to communicate among nodes on all subnetworks within this particular network.

Subnets, an example

Instead, if three bits are taken from the host field, eight networks can be formed, and the resulting network mask is as follows:

11111111.11111111.11111111.11100000

This subnet mask is 255.255.255.224. Each of the eight networks would have 30 nodes because five address bits are available. (It would be 32 except that all 1s, and all 0s are not legal host addresses.)

Network	Hosts	Address Range	Broadcast
192.168.1.0	192.168.1.0 to 192.168.1.30	192.168.1.30	
192.168.1.32	192.168.1.32 to 192.168.1.63	192.168.1.63	
192.168.1.64	192.168.1.64 to 192.168.1.95	192.168.1.95	
192.168.1.96	192.168.1.96 to 192.168.1.127	192.168.1.127	
192.168.1.128	192.168.1.128 to 192.168.1.159	192.168.1.159	
192.168.1.160	192.168.1.160 to 192.168.1.191	192.168.1.191	
192.168.1.192	192.168.1.192 to 192.168.1.223	192.168.1.223	
192.168.1.224	192.168.1.224 to 192.168.1.254	192.168.1.254	

TIP: If you need some help calculating subnets, check out the online subnet calculator at www.cotse.com/networkcalculator.html.

Communicating across networks: gateways and routing

Due to the structure of IP addressing, hosts can only communicate with other hosts on the

same network. To overcome this limitation, we add routing and gateways to the networking equation. Routing is the mechanism that determines the path a packet takes from its source to its destination. This path, or route, is established by looking up the destination IP address in a *routing table*. If the address is found, the packets are delivered to the network; if the address is not found, the packets are forwarded to an entry termed the *default route*, which is the IP where all addresses "unknown" to that machine or device are forwarded.

The machine or device that performs these routing and/or forwarding functions is called a gateway or router. Sometimes these two terms are used interchangeably. Technically speaking, a gateway describes a system or device that sends messages between different types of networks; a router sends messages between networks of the same type. We won't get too fussy in our discussions here given that we're only dealing with TCP/IP networking, but be aware that there is a difference between the two.

It's also important to note that gateways are, by definition, equipped with more than one network interface (say, A and B), each configured with a different IP or IP/subnet. This is how the "network bridging" effect is achieved. Packets arrive on interface A, and according to the entries in the device's routing table, one of four actions takes place. Packets can be:

- * Delivered to a host on network A
- * Passed to the default routing device "upstream" on network A
- * Delivered to a host on network B
- * Passed to the default routing device "upstream" on network B

Routing configurations

There are four common types of routing configurations:

- * *Minimal* -- A network completely isolated from all other networks requires only minimal routing. A minimal routing table is usually built when a network interface is initialized. If you have no need to communicate with other TCP/IP networks, and you are not using subnetting, this is all the routing information your system needs.
- * *Static* -- A network with one or two gateways is typically configured using static routing. A static routing table is automatically created by a network configuration script (using an IP supplied by the user) or manually by a system administrator (using the `/sbin/route` command discussed in Section 5). This table is as the name implies -- static. It does not adjust to network changes. When a network does change, the table must be manually re-configured.
- * *Dynamic* -- Large networks often have multiple routers and/or gateways installed, perhaps pointing to the same remote network for redundancy/failover purposes. This is the realm of dynamic routing. Dynamic routing allows routing tables to be dynamically constructed (and continuously updated) from information exchanged between routing devices using protocols called... yes, you guessed it... routing protocols. This continuous exchange of information provides a mechanism whereby routes automatically adjust to account for changing network conditions such as high traffic loads or outages. The biggest drawback to dynamic routing is that the route information exchanged between

devices can consume considerable bandwidth.

- * A fourth option is to use a combination of static and dynamic routing. Machines on each subnet use static routing to reach their immediate neighbors. The default route -- the route used from packets that are not assigned a specific route by the routing table -- is set to a gateway machine that's configured to provide dynamic routing and hence knows about networks beyond its own subnet.

Dynamic routing is based on one of several routing *protocols* (RIP, Hello, OSPF, etc. for interior protocols; EGP, BGP for exterior protocols), and is enabled via the gateway routing daemon, `gated`. Dynamic routing is beyond the scope of this tutorial. For more information, see Craig Hunt's *TCP/IP Network Administration* listed at the end of this tutorial in "Summary, resources, and feedback".

Coming up...

It's now time to take all the preceding theoretical knowledge and put it to use. In the coming sections we're going to look at:

- * The configuration files responsible for a network interface under Red Hat 7.0
- * Examining and configuring an interface using **`ifconfig`**
- * Adding and editing static routes
- * Using **`netstat`** to monitor/troubleshoot a network interface

Section 4. TCP/IP configuration files (Red Hat 7.0)

The key files and scripts

The actual process of initializing a network interface (sometimes referred to as "bringing an interface up") is controlled by a set of configuration files and scripts, most of which reside under the `/etc` directory. The configuration files tell Linux what its IP address, host name, and domain name are; the scripts are responsible for initializing the network interfaces.

Unfortunately, to date there is no agreed-upon standard for file locations and naming conventions across distributions. To provide concrete examples for this section, we'll base our descriptions on the popular Red Hat 7.0 package. Keep in mind that if your distribution is not Red Hat, or based on Red Hat conventions (like the Mandrake distributions), some of the files referenced here will be located in different directories or have different names. The net effect (pardon the pun) is the same, however -- pass the network information stored in a series of configuration files to a script, and, from the script, initialize the interface and network routes. The key files involved in initializing and configuring a network interface are:

- * `/etc/hosts` (Maps host names to IP addresses)
- * `/etc/networks` (Maps domain names to network addresses)
- * `/etc/sysconfig/network` (Sets networking on or off, the hostname, and the gateway)
- * `/etc/resolv.conf` (Sets the nameserver or DNS server IPs)
- * `/etc/rc.d/rc3.d/S10network` (Activates configured Ethernet interfaces at boot time, called by symlinks in the runlevel directories, `/etc/rc.d/rcN.d/`)
- * A collection of files in `/etc/sysconfig/network-scripts`. These include the main configurations for network connections, as well as symbolic links to provide interface status and control functions

`/etc/hosts`

`/etc/hosts` is a simple text file that associates IP addresses with host names. Every computer on a TCP/IP network must have a unique IP address. The hosts file simply allows users to relate a name to an IP address and use that name when accessing a computer rather than typing a string of numbers. Each entry in `/etc/hosts` contains an IP address separated by white space, followed by a host name and/or an alias. Comments begin with a hash (`#`). For example:

```
# /etc/hosts
# last updated 12/3/2000
127.0.0.1      loopback localhost      # loopback (lo0) name/address
192.168.1.5    janus.syroidmanor.com janus
192.168.1.6    thumper.syroidmanor.com thumper
192.168.1.7    donovan.syroidmanor.com donovan
192.168.1.8    raidserver
192.168.1.20   phoenix.syroidmanor.com phoenix
192.168.1.15   hydras.syroidmanor.com hydras
```

In the hosts file shown above, the IP address 192.168.1.5 is mapped to the host janus.syroidmanor.com, and is assigned an alternate host name (or alias) of janus. Although the host file has been superseded by DNS, it is still used for the following reasons:

- * Most systems have a small host table containing name and address information about key hosts on the local network. The table is used when DNS is not running, such as during initial system startup. Even if you have a local DNS server running, you should have a small hosts file on each system containing an entry for the host itself, an entry for *localhost*, and entries for any important gateways and servers on the LAN.
- * Small networks that are not connected to the Internet or other networks have no need for DNS services. They do, however, need to know how to locate their peers.

/etc/networks

Just as hosts have names and addresses, networks and subnets can also be named for convenience. The `/etc/networks` file is similar in layout to `/etc/hosts` with the name and address reversed.

```
# /etc/networks for syroidmanor.com
localnet      127.0.0.0      #loopback
syroid-C1     192.168.1      #development, class C
syroid-C2     192.168.2      #support, class C
```

In the example above, the network name *syroid-C1* can be used in scripts or with any command-line utility to reference the 192.168.1 class C network.

/etc/sysconfig/network

The `/etc/sysconfig/network` (note that network is singular, not plural, as opposed to the `/etc/networks` file) is used to specify information about the desired network configuration; it is used by several scripts at bootup. It can contain one or more of the following keyword/value pairings:

`NETWORKING=YES|NO` -- YES = networking should be configured; NO = it should not.

`HOSTNAME=hostname` -- the fully qualified domain name of the host; for compatibility with older programs, this should match the host entry in `/etc/hosts`.

`GATEWAY=gw-ip` -- the IP of the network's gateway.

`GATEWAYDEV=gw-dev` -- the name of the gateway device (for example, `eth0`).

`NISDOMAIN=dom-name` -- the NIS domain, if applicable.

The following is an example of a minimum `/etc/sysconfig/network` configuration:

```
NETWORKING=yes
HOSTNAME=phoenix.syroidmanor.com
GATEWAY=192.168.1.1
```

`/etc/resolv.conf` and `/etc/rc.d/rc3.d/S10network`

`/etc/resolv.conf` is one of the key files used by the network to determine host resolution. In it you can identify up to three nameservers; the last two provide backup if the first server listed fails to respond to a query. The domain entry defines the default domain name. The resolver (which, incidently, is not a separate process but a library of routines called by network processes) appends the domain name listed here to any hostname query that does not contain a dot.

```
# /etc/resolv.conf
# domain name resolver config file
domain syroidmanor.com
nameserver 192.168.1.7
nameserver 192.168.1.10
nameserver 165.142.268.19
```

In the example shown above, if a query is submitted to the resolver for host *phoenix* (note, no dot), the domain is appended to the request, which expands the query to *phoenix.syroidmanor.com*. For more details and available options, type **man resolv.conf**.

`/etc/rc.d/rc3.d/S10network` is a symlink to the `/etc/rc.d/init.d/network` script. It is responsible for initializing all configured network interfaces when the system reaches run-level 3. We won't take the time here to detail the logic behind this file as it primarily calls other scripts and programs referenced in this section. It's worth perusing, however, if you're interested in seeing the order various network components and services are initialized in (**less /etc/rc.d/rc3.d/S10network**).

The `/etc/sysconfig/network-scripts/` directory

Finally, the following files are normally found in the `/etc/sysconfig/network-scripts/` directory:

```
* /etc/sysconfig/network-scripts/ifup
* /etc/sysconfig/network-scripts/ifdown
* /etc/sysconfig/network-scripts/network-functions
* /etc/sysconfig/network-scripts/ifcfg-interface-name
* /etc/sysconfig/network-scripts/ifcfg-interface-name:clone-name
* /etc/sysconfig/network-scripts/chat-interface-name
* /etc/sysconfig/network-scripts/dip-interface-name
* /etc/sysconfig/network-scripts/ifup-post
```

In the following panels we'll look briefly at each of these key files, what they do, and what they contain.

.../network-scripts/, part 1

The `ifup` and `ifdown` entries in `/etc/sysconfig/network-scripts` are actually symbolic links to `/sbin/ifup` and `/sbin/ifdown`, respectively. These are the only two scripts in this directory that should be called directly, and they call all other scripts as needed.

`ifup` and `ifdown` take one argument normally: the name of the device (for example, `eth0`). They are called by the system with the argument "boot" during the boot process so that devices not configured to be initialized on system start are not activated (see `ONBOOT=no` under the interface-name description below).

`network-function` is not a public file. It contains functions required by several scripts within this directory. Specifically, it contains most of the code for handling alternative interface configurations.

.../network-scripts/, part 2

The configuration files `ifcfg-interface-name` and `ifcfg-interface-name:clone-name` contain the bulk of the details required to initialize an interface. The first file defines an interface, while the second file contains only parts of the definition pertinent to an "alias" (or alternative) interface. For example, the network address might be different, but everything else would be the same so only clone while would contain only an entry for `NETWORK=addr`.

Items defined in the `ifcfg` file depend on interface type; the following values are common:

- * `DEVICE=name`, where *name* is the name of the physical device
- * `IPADDR=addr`, where *addr* is the IP address
- * `NETMASK=mask`, where *mask* is the netmask value
- * `NETWORK=addr`, where *addr* is the network address
- * `BROADCAST=addr`, where *addr* is the broadcast address
- * `GATEWAY=addr`, where *addr* is the gateway address
- * `ONBOOT=answer`, where *answer* is "yes" (activate device on boot) or "no"
- * `USERCTL=answer`, where *answer* is "yes" (non-root users can control the device) or "no"
- * `BOOTPROTO=proto`, where *proto* is one of the following: "none" (no boot-time protocol), "bootp" (use the BOOTP protocol), or "dhcp" (use the DHCP protocol)

In addition, the following values are common to all SLIP (Serial Line IP) files:

- * `PERSIST=answer`, where *answer* is "yes" (keep the device active even if the modem has hung up) or "no" (do not keep active)
- * `MODEMPORT=port`, where *port* is the modem port's device name (for example, `/dev/modem`)
- * `LINESPEED=baud`, where *baud* is the modem's linespeed
- * `DEFABORT=answer`, where *answer* is "yes" (insert default abort strings when

creating/editing the script for this interface) or "no" (do not insert the default abort strings)

.../network-scripts/, part 3

The chat *-interface-name* file is a chat script for SLIP connections. Its function is to initiate the connection. For SLIP devices, a DIP script is written from the chat script.

The chat *-interface-name* is a write-only script created from the chat script by the program netcfg. DO NOT modify this file.

/etc/sysconfig/network-scripts/ifup-post is called when any network device (except a SLIP device) is initialized. It calls

/etc/sysconfig/network-scripts/ifup-routes to bring up static routes that depend on the device, brings up any aliases configured for the device, and sets the hostname if it is not already set -- and a hostname can be found for the IP matching the device. Finally, ifup-post sends a signal (SIGIO) to any programs that have requested notification of network events.

Section 5. Configuring network interfaces and routes

The ifconfig program

The `ifconfig` command sets, checks, or monitors configuration values for network interfaces. It can also be used to set the "state" of an interface -- that is, "up" (initialized) or "down" (uninitialized). A simple invocation of `ifconfig` is:

```
ifconfig interface-name ip-address up|down
```

This activates the specified interface and assigns the supplied IP address to it.

`ifconfig` has numerous options available (metric, mtu, pointtopoint, etc; see the man page for details) for explicitly setting unique interface parameters, but generally speaking, supplying the interface name (for example, `eth0`), IP address, and netmask are enough. For example:

```
ifconfig eth0 192.168.1.5 netmask 255.255.255.0 up
```

assigns interface `eth0` the IP 192.168.1.5, the netmask 255.255.255.0, and "brings it up" or initializes it. Similarly, to take the interface "down", type **`ifconfig eth0 down`**; IP and netmask need not be specified.

Using ifconfig to inspect an interface

Running `ifconfig` with no arguments causes the program to display the status of all network interfaces. To check the status of a specific interface, append the name after `ifconfig`. For example:

```
[tom@phoenix tom]$ /sbin/ifconfig eth0
eth0  Link encap:Ethernet  HWaddr 00:10:5A:00:87:22
       inet addr:192.168.1.20  Bcast:192.168.1.255  Mask:255.255.255.0
       UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
       RX packets:9625272 errors:0 dropped:0 overruns:0 frame:0
       TX packets:6997276 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:100
       Interrupt:19 Base address:0xc800
```

The above output shows the MAC address (Hwaddr), assigned IP address (inet addr), broadcast address (Bcast), and netmask (Mask). In addition, you can see the interface is UP with an MTU of 1500 and a Metric of 1. The next two lines give statistics on the number of packets received (RX) and transmitted (TX), along with packet error, dropped, and overrun counts. The last two lines shown the number of packet collisions, the transmit queue size (txqueuelen), and the IRQ and base address of the card.

Configuring routes

Let's begin our look at configuring routes by looking at a network interface that has no gateway configured. As you can see, using the route command without arguments displays the kernel routing table.

```
[root@phoenix tom]# /sbin/route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
127.0.0.0        127.0.0.1        255.0.0.0        U        0      0        0 lo
192.168.1.0      192.168.1.5      255.255.255.0    U        0      0        0 eth0
```

The first entry is the loopback route to *localhost* that was automatically created when *lo* was configured. The second entry is the route to network 192.168.1.0 through interface *eth0*. Address 192.168.1.5 is not a remote gateway address. It is the address assigned to the *eth0* interface on *phoenix*.

Note the flags for each entry. Both have the U (up) flag set, indicating they are ready to be used, but neither has the G (gateway) flag set. The G flag is not set because both of these routes are direct routes through local interfaces, not through external gateways.

The above example contains only one network route, 192.168.1.0. Therefore *phoenix* can only communicate with hosts located on that network.

Adding static routes

A minimal routing table only allows hosts on the same network to communicate. To reach remote hosts, routes through external gateways must be added to the routing table. One way to accomplish this is through the use of the `/sbin/route` command. Using the example from the previous panel, we're now going to add the route 192.168.1.1 to our network configuration.

```
[root@phoenix tom]# /sbin/route add default 192.168.1.1 1
```

The first argument after the route command in the above example is the keyword **add**. The first keyword on a route command is either **add** or **del** (delete). The next value is the destination address, which is the address reached via this route. If the keyword **default** is used for the destination address, a *default route* is created. The default route is used whenever there is no specific route to a destination; often this is the only entry you'll need in your routing table. If your network has only one gateway, use a default route to direct all traffic bound for remote networks through that gateway.

Next on the command line is the gateway address. The address must be the address of a gateway on a directly connected network. TCP/IP routes specify the *next-hop* in the path to the remote destination. That next-hop must be directly accessible to the local host; therefore, it must be on a directly connected network.

NOTE: Because most routes are added early in the system startup process, numeric IPs are recommended over host names. This is done to ensure the routing configuration is not

dependent on the state of the name server software. And always use complete numeric address (all four bytes); route guesses at partial IPs and you may end up with an incorrect configuration.

Static routes, continued

Referring to the route command on the previous panel, the last argument is the number 1; it is called the *routing metric*. The metric argument is not needed when a route is deleted, but many systems require it when a route is added. Despite being required, route only uses the metric to decide if the route is through a directly attached interface or through an external gateway. If the metric is 0, the route is installed through a local interface and the G flag is not set; if the metric value is greater than 0, the route is installed with the G flag and the gateway address is assumed to be external. Static routing makes no other use of the metric. Dynamic routing is needed to make real use of varying metric values.

To display our new routing table, type `/sbin/route` or use the `netstat -rn` command (which we'll discuss in the next section):

```
[root@phoenix tom]# netstat -rn
```

```
Kernel IP routing table
```

Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
192.168.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	192.168.1.1	0.0.0.0	UG	0	0	0	eth0

As a final test that everything is working as advertised, ping a host on another network; you should receive a response. If you don't, recheck your configuration.

To familiarize yourself with other route options and arguments, type **man route**.

Section 6. Monitoring your TCP/IP network

The netstat program

If you manage a TCP/IP network of any size, the `netstat` program is an invaluable tool. It can display the kernel routing table, the status of active network connections, and useful statistics for each installed network interface.

Like most Linux administrative command-line programs, the amount of detail and/or range of information `netstat` provides is selectable by appending options or flags to the program command. Some of the more common options are:

- a -- shows information on all connections, including those just listening
- i -- shows statistics for all configured network devices
- c -- continually updates the network status (once per second) until interrupted(^C)
- r -- displays the kernel routing table
- n -- shows remote and local addresses in numeric (raw) format rather than resolved names
- t -- shows only TCP socket information (excluding any UCP socket information)
- v -- displays the version information for netstat

Type `man netstat` for a complete list of available flags and a detailed explanation of what each one does. Note that you can also combine flags, so entering `netstat -rn` displays the system routing table (r) in raw IP format for local and remote hosts (n).

Displaying active network connections

`netstat` supports a set of options to display active or passive sockets: -t, -u, -w, and -x show active TCP, UDP, RAW, or UNIX socket connections, respectively. If the -a flag is added, sockets that are waiting for a connection (in other words, listening) are shown as well. This display will give you a list of all servers that are currently running on your system.

For example, typing `netstat -ta` on the host *phoenix* produces the following:

```
[tom@phoenix tom]$ netstat -ta
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address   Foreign Address   State
tcp    0      0 phoenix.syroidmanor:ssh 192.168.1.5:1132   ESTABLISHED
tcp    0      0 *:ssh           *:                LISTEN
tcp    0      0 phoenix.syroidmano:1028 hydras.syro:netbios-ssn ESTABLISHED
tcp    0      0 phoenix.syroidmano:1027 raidserver:netbios-ssn ESTABLISHED
tcp    0      0 *:printer       *:                LISTEN
tcp    0      0 *:auth          *:                LISTEN
tcp    0      0 *:1024          *:                LISTEN
tcp    0      0 *:sunrpc        *:                LISTEN
```

The above output shows most servers simply waiting for an incoming connection (LISTEN). The first line, however, shows a connection between the host phoenix and the IP address 192.168.1.5; the third and fourth lines show two netbios connections (Samba SMB shares).

Viewing the routing table with netstat

When invoked with the **-r** flag, netstat displays the kernel routing table similar to typing **/sbin/route**:

```
[tom@phoenix tom]$ netstat -nr
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
127.0.0.0 0.0.0.0 255.0.0.0 U 0 0 0 lo
0.0.0.0 192.168.1.1 0.0.0.0 UG 0 0 0 eth0
```

The **-n** option forces netstat to output addresses as dotted quad IP numbers rather than the symbolic host and network names. This option is especially useful when you want to avoid address lookups over the network (to a DNS or NIS server, for example).

The second column displays the gateway the routing entry points to. If no gateway is used, an asterisk is shown instead. The third column is the netmask for the route. The kernel uses this to set the "generality" of a route by bitwise ANDing the Genmask against a packet's IP address before comparing it to the destination IP address of the route.

The fourth column displays flags for the route: U means up, H means host, G means gateway, D means dynamic route, and M means modified.

Viewing the routing table with netstat, continued

```
[tom@phoenix tom]$ netstat -nr
Kernel IP routing table
Destination Gateway Genmask Flags MSS Window irtt Iface
192.168.1.0 0.0.0.0 255.255.255.0 U 0 0 0 eth0
127.0.0.0 0.0.0.0 255.0.0.0 U 0 0 0 lo
0.0.0.0 192.168.1.1 0.0.0.0 UG 0 0 0 eth0
```

The next three columns show the *MSS*, *Window*, and *irtt* that will be applied to TCP connections established via this route. The MSS is the Maximum Segment Size and is the size of the largest datagram the kernel will construct for transmission via this route. The Window is the maximum amount of data the system will accept in a single burst from a remote host.

The acronym irtt stands for "initial round trip time." The TCP protocol ensures that data is reliably delivered between hosts by retransmitting a datagram if it has been lost. The TCP protocol keeps a running count of how long it takes for a datagram to be delivered to the remote end, and an acknowledgement to be received so that it knows how long to wait before assuming a datagram needs to be retransmitted; this process is called the round-trip time. The

initial round-trip time is the value that the TCP protocol will use when a connection is first established. For most network types, the default value is okay, but for some slow networks, notably certain types of amateur packet radio networks, the time is too short and causes unnecessary retransmission. The `irtt` value can be set using the `route` command. Values of zero in these fields mean that the default is being used.

Finally, the last field displays the network interface that the displayed route will use.

Displaying usage statistics with `netstat`

Invoking `netstat` with the `-i` option displays usage statistics for all configured interfaces -- an excellent tool for troubleshooting network problems. With this command, it's easy to check on both the status and the "health" of connections.

```
[tom@phoenix tom]$ netstat -i
Kernel Interface table
eth0      Link encap:Ethernet  HWaddr 00:10:5A:00:87:22
          inet addr:192.168.1.20  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10554374 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8528339 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:19 Base address:0xc800
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:5612 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5612 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

The `RX packets` and `TX packets` lines show how many packets have been received or transmitted, respectively, as well as RX/TX statistics for errors, dropped packets, and overruns. The most common interface errors stem from incorrect configuration, so if you're experiencing difficulties, always start your diagnosis by triple checking all settings.

Providing the interface is up, there should be no packets queued for transmission (`txqueuelen`) -- if there are, suspect a bad network cable or network card. Start by swapping in a spare cable and rechecking the connection. RX/TX errors should be close to zero. High TX errors could indicate a saturated network or a bad physical connection; high RX errors could indicate a saturated network, a bad physical connection, or an overworked host. If you experience high collision rates (collision rates are a percent of *output packets*, and are not calculated from the total number of packets sent/received) it could also be indicative of a saturated network; confirm this by running `netstat -i` from another host on the same subnet and comparing results.

To troubleshoot network errors, careful and methodical analysis of all aspects of the interface (hardware and software) is essential. Don't get in a rush, and... ahem... did we mention always check your network cable first? Trust us on this one.

Section 7. Summary, resources, and feedback

Summary

In this tutorial we've discussed the history of TCP/IP, the OSI model and its relationship to TCP/IP's design, IP addressing, subnetting, and routing -- all from a theoretical perspective. We then rolled up our sleeves and examined how a TCP/IP network is initialized under Red Hat 7.0 and which files do what. Next came a look at how to configure a network interface, and how to designate a route between your local LAN and the "outside world". The tutorial finished up with a look at the `netstat` program, and how to use it to examine the health of your network.

That's definitely lot of material to absorb, but TCP/IP is a huge topic with hundreds of sideroads one can wander down and get lost on for a day or three. The facts remain: TCP/IP is the backbone of the Internet, routing is the glue that holds it all together, and IP addressing represents the places we visit as we surf the world. And when you think about it, given the explosive growth demands that the Internet in particular, and computing general have seen over the last three or four years, the infrastructure really is holding up remarkably well.

Resources

On the Web:

- * [Linux System Administrator's Guide](#)
- * Learn how to [Easily configure TCP/IP on your AIX system](#)
- * IBM offers tools for network monitoring such as the [Tivoli NetView Performance Monitor for TCP/IP](#)
- * Visit the home page for [TCP/IP for OS/400](#)
- * Read about [Implementing TCP/IP and Internet access for iSeries and AS/400](#)
- * If online research and reference are your thing, I highly recommend O'Reilly's new [Safari subscription service](#). You get a fully searchable selection of some of the best networking books ever written.

In print:

TCP/IP Network Administration, 2nd Edition, Craig Hunt (O'Reilly) ISBN: 1-56592-322-7
Linux in a Nutshell, 3rd Edition, Siever, Spainhour, Figgins, and Hekman (O'Reilly) ISBN: 0-596-00025-1
Running Linux, 3rd Edition, Welsh, Dalheimer, and Kaufman (O'Reilly) ISBN: 1-56592-469-X

Your feedback

We look forward to getting your feedback on this tutorial. Additionally, you are welcome to contact the author, Tom Syroid, directly at tom@syroidmanor.com.

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

You can get the source code for the Toot-O-Matic at www6.software.ibm.com/dl/devworks/dw-tootomatic-p. The tutorial [Building tutorials with the Toot-O-Matic](#) demonstrates how to use the Toot-O-Matic to create your own tutorials. developerWorks also hosts a forum devoted to the Toot-O-Matic; it's available at www-105.ibm.com/developerworks/xml_df.nsf/AllViewTemplate?OpenForm&RestrictToCategory=11. We'd love to know what you think about the tool.