Capítulo 11: Conceptos fundamentales sobre redes (Tema 109)

Sin una forma de encaminar el tráfico de datos de un lugar a otro, la comunicación en red (lo que incluye a Internet) sería simplemente imposible. Esto significa que las maravillas que hoy damos por hechas (el correo electrónico, los sitios web, el streaming de audio y video, por nombrar algunos ejemplos) no existirían. En este capítulo repasaremos los protocolos de transmisión (TCP/IP, UDP, y ICMP), la manera de administrar direcciones de red y tablas de enrutamiento, y el concepto de puertos de servicio.

Protocolos

Los protocolos TCP/IP fueron adoptados como estándar a principios de la década de 1980. Desde entonces, se han convertido en la base sobre la que se realizan las comunicaciones en todo tipo de redes sin importar su tamaño o complejidad.

El desafío que se generó a partir de la necesidad de unir equipos remotos no fue solamente a nivel técnico, sino también a nivel comercial ya que cada empresa intentó imponer su norma en la industria. El desarrollo de estándares fue importante y necesario, ya que fue la piedra fundamental para que dos equipos remotos (inclusive habiendo sido fabricados por distintas empresas) hablaran el mismo idioma y de esa manera pudieran comunicarse.

Cada uno de estos estándares se divide en capas, y cada una de estas tiene un propósito específico. El modelo de capas que vamos a adoptar es el modelo TCP/IP de 4 capas, que es esencialmente el modelo OSI, pero más simplificado para lo que es propiamente *networking*:

TCP/IP
Capa de aplicación
Capa de transporte
Capa de Internet
Capa de Acceso a las Redes (NAL)

Modelo OSI
Capa de aplicación
Capa de presentación
Capa de sesión
Capa de transporte
Capa de red
Capa de enlace de datos
Capa física

Veamos una descripción de cada una de estas capas:

- La **capa 1** o **capa de acceso a la red** especifica las características del hardware (incluyendo el medio de transporte) y la forma en la que los datos deben enrutarse.
- La **capa 2** o **capa de Internet** permite administrar las direcciones de enrutamiento de datos. Básicamente permite que un paquete llegue desde un origen a un destino. Esta capa engloba protocolos como **IP** (**I**nternet **P**rotocol) e **ICMP** (**I**nternet **C**ontrol **M**essage **P**rotocol).
- La **capa 3** o **capa de transporte** se encarga de transporte de datos, su división en paquetes y la administración de potenciales errores de transmisión. Es aquí donde encontramos al protocolo **TCP** (**T**ransmission **C**ontrol **P**rotocol).
- La **capa 4** o **capa de aplicación** es administrada directamente por el software y está relacionado con el tipo de servicio que provee una determinada aplicación. Ejemplo de protocolos de esta capa son **ssh**, **http**, **ftp**, y **rsyslog**. Cada uno de estos protocolos tiene una finalidad específica (**http** es para la web, **ftp** para transferir archivos, etc).

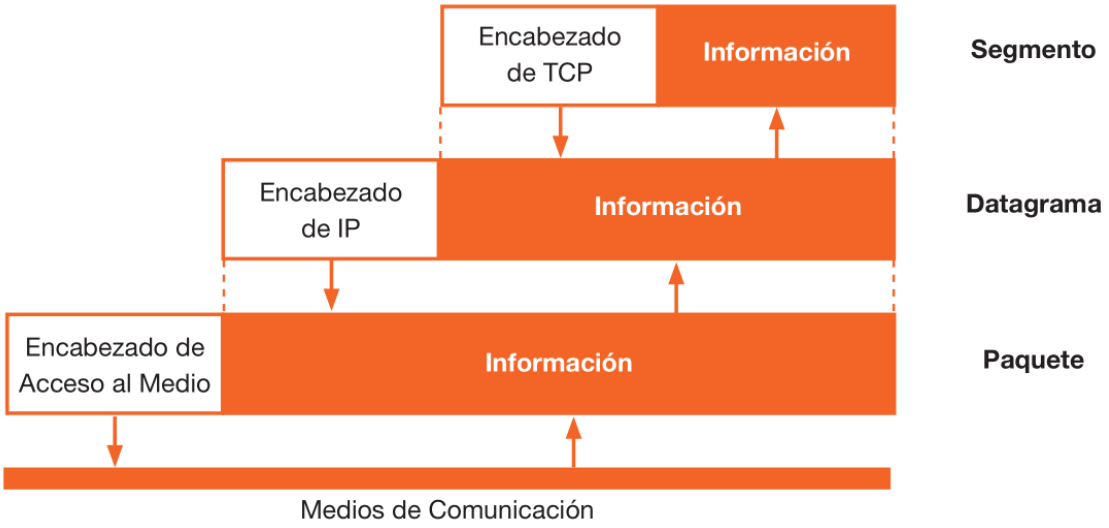
Como podemos apreciar, capa implementa uno o más protocolos y tiene un propósito específico.

Paquetes de red y encapsulamiento de datos

La información se divide en paquetes que viajan a través de los nodos o **routers** y se reensamblan en el destino final. El esquema de transmisión de datos se llama **routing** (enrutamiento) y los datos o paquetes de datos son normalmente conocidos como datagramas. Genéricamente podemos decir que cada paquete se compone de:

- Una **cabecera** o **encabezado**, que es específica de cada protocolo y tiene información para cumplir el objetivo de esa capa.
- Un **área de datos** (también llamada *payload*) que contiene los datos que se desean transmitir.
- Una **cola**, que generalmente contiene información para la detección de errores.

Cuando se transmite de un origen a un destino, los datos atraviesan las capas en el equipo remitente. En cada una de esas capas se le coloca al principio del paquete de datos información que conocemos como **encabezado**. A su vez, en el equipo receptor cuando atraviesa cada capa, el encabezado correspondiente a cada una se lee, se corrobora la información, elimina ese encabezado y envía el paquete hacia la capa superior.



En resumen, para ilustrar este concepto podríamos trazar una analogía con una cebolla (por las capas), o con las mamushkas rusas.

El protocolo IP

Los datagramas como unidades más pequeñas de la transmisión de datos son intercambiados por el protocolo IP, que es completamente independiente del hardware. La ventaja principal de este protocolo es que permite unir redes físicamente separadas en una red aparentemente homogénea.

Las funciones principales del protocolo IP son:

1. **Definir datagramas**. Cuando enviamos un archivo vía red, el protocolo lo divide en partes más pequeñas, es decir, bloques de datos o paquetes.
2. **Establecer la dirección en la red** en el encabezado, junto con la identificación del equipo que envió los paquetes.
3. **Routing de datagramas a computadoras lejanas**. Si un datagrama va dirigido a una computadora que no se encuentra en la misma red, es direccionado a un *gateway* (puerta de enlace) para llegar al destino indicado.

---

*Es importante aclarar que el protocolo IP no dispone de información de control de la transmisión o **handshake**. En otras palabras, transporta los paquetes de un lugar a otro sin un mecanismo de verificación que asegure que los paquetes se recibieron en el orden correcto.*

A continuación, hablaremos de las dos versiones de este protocolo que se encuentran actualmente en uso.

IPv4

Una dirección **IPv4** consiste en cuatro octetos con números entre 0 y 255 cada uno (**192.168.0.100**, por ejemplo) donde la primera parte identifica la red y la segunda un equipo en particular. Cada uno de estos números está formado por 8 bits (de ahí el nombre de *octeto*), por lo que tiene una limitación de 2^32 (4.294.967.296) direcciones posibles. Si bien este número es gigantesco, la proliferación de dispositivos móviles e inteligentes en la última década ocasionará que en el futuro cercano se agoten las direcciones disponibles.

Estrictamente hablando, las direcciones IP se dividieron en las siguientes clases desde 1981 hasta mediados de la década de 1990 cuando se introdujo la **CIDR** (**C**lass-less **I**nter-**D**omain **R**outing). Sin embargo, es común todavía continuar utilizando estas categorías:

- **A**: 0.0.0.0 a 127.255.255.255 con máscara de red igual a 255.0.0.0. En esta clase, el primer octeto se utiliza para indicar la red, mientras que los tres restantes están disponibles para equipos. Por nombrar un ejemplo, una red con dirección 10.0.0.0 y máscara de red 255.0.0.0 puede expresarse en notación CIDR como 10.0.0.0/8.
- **B**: 128.0.0.0 a 191.255.255.255 con máscara de red igual a 255.255.0.0 (dos octetos para representar la red y los dos restantes para equipos). Una red con dirección IP 172.19.0.0 y máscara 255.255.0.0 puede expresarse en notación CIDR como 172.19.0.0/16.
- **C**: 192.0.0.0 a 223.255.255.255 con máscara de red igual a 255.255.255.0 (tres octetos para representar la red y el restante para equipos). Una red con dirección IP 192.168.0.0 y máscara 255.255.255.0 puede expresarse en notación CIDR como 192.168.0.0/24.

---

*Las direcciones que terminan en **.0** y en **.255** representan la dirección de la red y la de broadcast, respectivamente, no pueden asignarse a ningún equipo en particular.*

- **D**: 224.0.0.0 a 239.255.255.255 (broadcast)
- **E** (reservada): 240.0.0.0 a 255.255.255.255

Sin importar el tipo de red, los mensajes enviados a la dirección de broadcast son recibidos por todos los hosts de la red. La máscara nos permite indicar a qué red pertenece una determinada dirección IP.

A pesar de lo útiles que fueron en su momento, la especificación de direcciones IP mediante clases cedió ante la introducción de CIDR. Este último método permite la creación de subredes (llamadas comúnmente *subnets*) tales como 192.168.0.0/25. En este caso, los primeros 25 bits representan la red (con una máscara igual a 255.255.255.128) y los últimos 7 las direcciones disponibles para los equipos (desde **.1** hasta **.126**, siendo **.127** la dirección de broadcast).

Como podemos ver, CIDR hace posible diseñar redes del tamaño que se adapte mejor a nuestras necesidades, y facilita las tareas de administración.

IPv6

Esta versión del protocolo fue definida en 1995 por la **I**nternet **E**ngineering **T**ask **F**orce (**IETF**).

Una dirección IPv6 es un número hexadecimal de 128 bits distribuidos en 16 octetos (por ejemplo, **2DAF:FF40:0928:CD01:4433:00DD:0988:FFFF**), aumentando la cantidad de direcciones disponibles a 2^128. Además de superar la limitación de IPv4 en este aspecto, esta versión del protocolo incluye mejoras en la seguridad y de rendimiento en la transferencia de paquetes de datos.

TCP

Como hemos mencionado antes, el protocolo IP no dispone de control de la transmisión. Por este motivo, generalmente se habla de la dupla **TCP/IP**, ya que es **TCP** el que se ocupa de la seguridad del envío. Este último es un protocolo de flujo de datos (también llamado *byte stream*), presenta confiabilidad y es orientado a la conexión. Veamos en detalle estas características:

- **Flujo de datos**: TCP considera los datos como una unidad de paquetes no separados, en vez de una secuencia de paquetes independientes.
- Provee **confiabilidad** porque verifica si todos los datagramas que se enviaron han llegado a destino. ¿Cómo lo hace? En primer lugar, se asegura de que hayan llegado todos los paquetes. Si no recibe confirmación del receptor, obliga a una retransmisión de los paquetes necesarios hasta que todos hayan sido recibidos con éxito.
- **Orientado a la conexión** significa que TCP establece una conexión lógica entre dos equipos. Antes de la transmisión de los datagramas, TCP transmite informaciones de control (*handshake*) entre el emisor y el receptor iniciando la comunicación entre los dos *hosts*.

Por las razones que acabamos de nombrar, las aplicaciones que se comunican a través de TCP no necesitan implementar sus propios mecanismos de detección de errores (ya que el propio protocolo se encarga de esa tarea).

ICMP y UDP

Estos dos protocolos comparten la característica de **no** ser orientados a la conexión. UDP provee acceso directo a IP, facilitando el intercambio de información, pero sin la carga extra que ocasiona el *handshake*. La única desventaja es que las aplicaciones que se comunican mediante este protocolo deben encargarse de chequear errores de transmisión, ya que UDP no provee un método para confirmar que los paquetes hayan llegado a su destino correctamente. Dos aplicaciones conocidas que utilizan UDP son DNS y NFS.

ICMP, por otro lado, se utiliza para intercambiar información de control entre equipos. En el caso de que el equipo destino no pueda procesar el tráfico entrante, puede enviar mensajes mediante ICMP informando sobre la situación. Cuando se hace una solicitud para un host que no se puede alcanzar por algún motivo (ya sea que no esté disponible o que no se haya especificado una ruta para alcanzarlo), se utiliza ICMP para indicar el problema. También se utiliza el mismo recurso para reportar cuando el equipo remoto responde correctamente.

```
alumno@debiancla:~$ ping -c 4 www.google.com
PING www.google.com (216.58.222.36) 56(84) bytes of data.
64 bytes from gru09s17-in-f36.1e100.net (216.58.222.36): icmp_seq=1 tt
64 bytes from gru09s17-in-f36.1e100.net (216.58.222.36): icmp_seq=2 tt
64 bytes from gru09s17-in-f36.1e100.net (216.58.222.36): icmp_seq=3 tt
64 bytes from gru09s17-in-f36.1e100.net (216.58.222.36): icmp_seq=4 tt

--- www.google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3005ms
rtt min/avg/max/mdev = 22.554/23.264/23.847/0.527 ms
alumno@debiancla:~$ ping -c 4 123.123.123.123
PING 123.123.123.123 (123.123.123.123) 56(84) bytes of data.
^C
--- 123.123.123.123 ping statistics ---
4 packets transmitted, 0 received, 100% packet loss, time 3051ms

alumno@debiancla:~$
```

Como apreciamos en la imagen de arriba, un ejemplo de este último caso es el uso del comando `ping` para comprobar la disponibilidad de un host.

Servicios de red

Debido a que un mismo equipo puede ofrecer varios servicios de red, el tráfico entrante necesita ser direccionado al puerto correcto. Por convención, los puertos 1 al 65535 se dividen en los siguientes tres grandes grupos:

- 1 al 1023: puertos para servicios conocidos. En este rango encontramos al 21 (FTP), 22 (SSH), 23 (Telnet), 25 (SMTP), 53 (DNS), 80 (HTTP), 443 (HTTPS), 123

(NTP), 110 (POP3), 143 (IMAP), 993 (IMAPS), y 995 (POP3S), por nombrar algunos ejemplos.

---

*Para dirigir una solicitud a un puerto específico se debe indicar el destino mediante la combinación **IP:puerto**. De esta forma, una petición al puerto 443 (HTTPS) de 192.168.0.170 tendrá la forma 192.168.0.170:443.*

- 1024 al 49151: puertos registrados por ICANN (reservados para protocolos comerciales)
- 49152 al 65535: puertos disponibles.

En el archivo **/etc/services** se puede encontrar una lista completa y actualizada de la lista de servicios y sus puertos utilizados por defecto.

Configuración persistente de interfaces de red

Antes de que nuestro equipo pueda trabajar en red, es necesario que disponga de al menos una interfaz de red configurada. En el caso de Debian y derivados esta tarea se lleva a cabo a través del archivo **/etc/network/interfaces**. Por otro lado, en CentOS y similares se utilizan archivos individuales ubicados dentro de **/etc/sysconfig/network-scripts** para cada interfaz por separado.

A continuación, podemos ver la configuración estática de enp0s3 a través de la directiva `iface enp0s3 inet static`. Bajo esta modalidad, asignamos una dirección IP fija (192.168.0.170) a dicha interfaz:

```
# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet static
        address 192.168.0.170
        netmask 255.255.255.0
        gateway 192.168.0.1
        dns-nameservers 8.8.8.8 8.8.4.4
```

Luego, mediante las directivas **netmask** indicamos que esta interfaz se encontrará en la red 192.168.0.0/24. Finalmente, el gateway por defecto será 192.168.0.1 (generalmente, la dirección IP de nuestro router) y utilizaremos el servicio de DNS de Google (servidores con direcciones 8.8.8.8 y 8.8.4.4).

En este mismo archivo podemos colocar las configuraciones de todas las interfaces. La otra opción es hacerlo en archivos individuales dentro de **/etc/network/interfaces.d**, aunque la primera es la más común.

En CentOS, la forma típica de configurar las interfaces de red consiste en editar los archivos dentro de **/etc/sysconfig/network-scripts**. Por ejemplo, para configurar la interfaz **enp0s3** utilizaremos **/etc/sysconfig/network-scripts/ifcfg-enp0s3**. Mediante las siguientes líneas realizamos la misma configuración que en el caso anterior:

```
BOOTPROTO="static"
IPADDR=192.168.0.170
NETMASK=255.255.255.0
NETWORK=192.168.0.0
GATEWAY=192.168.0.1
DNS1=8.8.8.8
DNS2=8.8.4.4
ONBOOT="yes"
NM_CONTROLLED=no
```

Para aplicar los cambios realizados en cualquiera de los dos casos, recordemos reiniciar el servicio de red:

```
systemctl restart network # CentOS
systemctl restart networking # Debian
```

Además, en los siguientes archivos se almacenan otras configuraciones de red:

- **/etc/hosts** contiene relaciones entre direcciones IP y nombres de equipo. En redes pequeñas se utiliza como método de resolución de nombres.
- **/etc/nsswitch.conf**, entre otras cosas, controla el orden en que se consultan los recursos de resolución de nombres.
- **/etc/resolv.conf** especifica los servidores de nombres a utilizar por defecto en el sistema.

---

*Bajo **systemd**, existe un servicio llamado **systemd-resolved** que provee resolución de nombres a las aplicaciones locales.*

En la siguiente imagen podemos ver las líneas relevantes de los archivos que acabamos de mencionar:

```
alumno@debiancla:~$ head -n 2 /etc/hosts
127.0.0.1       localhost
127.0.1.1       debiancla
alumno@debiancla:~$ grep hosts /etc/nsswitch.conf
hosts:          files dns
alumno@debiancla:~$ cat /etc/resolv.conf
nameserver 8.8.8.8
nameserver 8.8.4.4
alumno@debiancla:~$
```

En el archivo **/etc/hosts** podemos ver que la dirección 127.0.0.1 ha sido relacionada con los nombres de equipo **localhost** y **debiancla**. De esta manera, al hacer

```
ping -c 4 debiancla
```

se enviarán 4 paquetes de verificación a la dirección 127.0.0.1. Lo mismo sucedería si utilizáramos la designación **localhost** en vez de **debiancla**.

Por otro lado, la línea que hemos resaltado de **/etc/nsswitch.conf**:

```
hosts:          files dns
```

indica que para la resolución de nombres de hosts se debe buscar primero en el archivo **/etc/hosts** (**files**) y luego en los servidores de nombres (**dns**). Una consecuencia directa de esto es que si hay dos equipos con nombre idéntico pero con direcciones IP distintas, tomará precedencia el que se encuentre presente en **/etc/hosts**. Si deseamos invertir el orden, podemos editar el archivo y cambiar la línea en cuestión por

```
hosts:          dns files
```

Finalmente, en **/etc/resolv.conf** se especifica que, por defecto, se utilizarán los DNS de Google para cualquier aplicación del sistema que necesite utilizar el servicio de resolución de nombres.

Herramientas esenciales

Afortunadamente, la línea de comandos nos provee un amplio arsenal de utilidades para realizar todo tipo de tareas de comprobación, configuración, y detección de errores de red. En las secciones que siguen repasaremos el uso de las más conocidas.

Comprobaciones con traceroute

Si al utilizar `ping` (o su equivalente `ping6` para IPv6) nos encontramos con que un equipo remoto no responde podemos utilizar `traceroute` (o `traceroute6`) para intentar identificar el punto donde se produce el inconveniente. Esta herramienta

devuelve el listado de las rutas seguidas en la transmisión de 3 paquetes por salto (incluyendo *ida y vuelta*) a un destino dado:

```
traceroute www.google.com
```

```
alumno@debiancla:~$ traceroute www.google.com
traceroute to www.google.com (172.217.28.196), 30 hops max, 60 byte packets
 1  192.168.0.1 (192.168.0.1)  0.856 ms  0.737 ms  1.237 ms
 2  * * *
 3  10.25.1.1 (10.25.1.1)  14.874 ms  15.658 ms  15.625 ms
 4  192.168.235.13 (192.168.235.13)  28.928 ms  29.479 ms  31.279 ms
 5  190.94.176.190 (190.94.176.190)  31.175 ms  31.812 ms  31.814 ms
 6  108.170.248.241 (108.170.248.241)  32.513 ms  28.279 ms 108.170.248.225 (10
 7  216.239.62.89 (216.239.62.89)  30.301 ms 216.239.62.87 (216.239.62.87)  29.
 8  eze03s30-in-f4.1e100.net (172.217.28.196)  34.088 ms  33.010 ms  33.684 ms
alumno@debiancla:~$
```

donde la presencia de * en la salida indica que el equipo en cuestión superó el tiempo de respuesta o está configurado para bloquear este tipo de solicitudes. Por defecto, `traceroute` utiliza ICMP pero puede emplear TCP si se acompaña el comando anterior con la opción `-T` (en este último caso será necesario disponer de privilegios administrativos).

Uso de ifconfig y route

Además de la configuración persistente de interfaces de red que explicamos en este capítulo, también es posible hacer cambios temporarios en un sistema en funcionamiento que desaparezcan con el próximo reinicio.

El comando `ifconfig` nos permite ver y actualizar la configuración de una interfaz de red. Por otro lado, `route` hace posible que modifiquemos la tabla de enrutamiento a fin de indicar los gateways o puertas de enlace que permitan alcanzar otras redes.

Al ejecutar ifconfig y route sin opciones ni argumentos veremos la lista de interfaces de red disponibles y la tabla de enrutamiento actual del kernel:

```
ifconfig
route
```

```
root@debiancla:~# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.170  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::a00:27ff:fedf:604b  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:df:60:4b  txqueuelen 1000  (Ethernet)
        RX packets 5560  bytes 509794 (497.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 1715  bytes 262124 (255.9 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 10  bytes 774 (774.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 774 (774.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@debiancla:~# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         192.168.0.1     0.0.0.0         UG    0      0        0 enp0s3
link-local      0.0.0.0         255.255.0.0     U     1000   0        0 enp0s3
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 enp0s3
root@debiancla:~#
```

Entre otras cosas, la imagen de arriba nos informa lo siguiente:

- Tanto **enp0s3** como **lo** están disponibles (UP).
- Las direcciones IPv4 e IPv6 de **enp0s3** son 192.168.0.170 y fe80::a00:27ff:fedf:604b, mientras que para el caso de **lo** son 127.0.0.1 y ::1 (donde todos los octetos faltantes son iguales a 0), respectivamente.
- La dirección física de **enp0s3** (más conocida como *MAC address*) es 08:00:27:df:60:4b. Los tres primeros grupos de caracteres hexadecimales (08:00:27) representan el fabricante de la placa.
- La puerta de enlace por **default** (donde se direcciona el tráfico sin un destino especificado o fuera de la red local) está localizada en 192.168.0.1 y se alcanza a través de enp0s3.
- Para alcanzar equipos de la red 192.168.0.0/24 (máscara igual a 255.255.255.0) se dispone de acceso directo.

Algunas operaciones que podríamos llegar a necesitar realizar son las siguientes:

Cambiar la IP a una interfaz (**¡CUIDADO!** Este procedimiento puede interrumpir nuestra conexión actual, por lo que debemos tomar los recaudos necesarios para revertir el cambio luego). Para establecer 192.168.0.150 como la dirección de enp0s3 haremos

```
ifconfig enp0s3 192.168.0.150
```

Modificar el gateway por defecto. A fin de especificar 192.168.0.102 como nuestra nueva puerta de enlace predeterminada podemos hacer

```
route add default gw 192.168.0.102
```

Agregar una ruta para una red particular. Por ejemplo, el comando

```
ip route add 192.168.1.0/24 via 192.168.0.254 dev enp0s7
```

hará que el tráfico destinado a la red 192.168.1.0/24 se encamine a través de enp0s7 hacia el equipo con direcicón IP igual a 192.168.0.254. Por supuesto, esto requiere que dicho sistema esté configurado para enrutar paquetes hacia dicha red.

Habilitar o deshabilitar una interfaz. Para desactivar **lo** temporariamente:

```
ifconfig lo down
```

Luego podemos activarla de nuevo con

```
ifconfig lo up
```

```
root@debiancla:~# ifconfig lo down
root@debiancla:~# ifconfig lo
lo: flags=8<LOOPBACK>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1  (Local Loopback)
        RX packets 10  bytes 774 (774.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 774 (774.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  co

root@debiancla:~# ifconfig lo up
root@debiancla:~# ifconfig lo
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1  (Local Loopback)
        RX packets 10  bytes 774 (774.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 10  bytes 774 (774.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  co

root@debiancla:~#
```

En la imagen de arriba se puede apreciar los últimos cambios que hicimos sobre **lo**.

El grupo de herramientas ip

Si bien `ifconfig` y `route` no han quedado obsoletos como a veces se dice, desde hace varios años su uso ha sido reemplazado por el grupo de herramientas `ip`. Este último comando incluye toda la funcionalidad de los anteriores simplemente modificando argumentos. Por ejemplo:

- Ver interfaces de red: `ip link show`
- Mostrar direcciones: `ip address show`
- Examinar la tabla de enrutamiento: `ip route show`
- Establecer dirección IP para una interfaz: `ip address add 192.168.0.170/24 dev enp0s3`
- Cambiar la puerta de enlace predeterminada a 192.168.0.1: `ip route add default via 192.168.0.1 dev enp0s3`

Para revertir lo que agregamos con `add` disponemos de la acción `del`. De esta forma,

```
ip address del 192.168.0.150 dev enp0s3
```

eliminará la dirección indicada de enp0s3, por ejemplo.

- También podemos deshabilitar **lo** con

```
ip link set dev lo down
```

y activarla posteriormente utilizando

```
ip link set dev lo up
```

Es importante destacar que estas son las operaciones más usuales. Para más información o detalles podemos recurrir al *man page* de `ip`, donde está disponible una lista más completa.

Los comandos host y dig

Tanto `host` como `dig` son dos herramientas que permiten hacer consultas a servidores DNS. La primera es más concisa y especialmente apta para su uso en aplicaciones que no requieran intervención del administrador (como, por ejemplo, un script), mientras que la segunda provee mayor cantidad de información detallada sobre una consulta en particular.

Para ilustrar, hagamos

```
host www.google.com
dig www.google.com
```

```
root@debiancla:~# host www.google.com
www.google.com has address 216.58.222.36
www.google.com has IPv6 address 2800:3f0:4002:807::2004
root@debiancla:~# dig www.google.com

; <<>> DiG 9.10.3-P4-Debian <<>> www.google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 62767
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONA

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.google.com.                        IN      A

;; ANSWER SECTION:
www.google.com.         223     IN      A       172.217.30.228

;; Query time: 44 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Oct 31 11:06:43 -03 2018
;; MSG SIZE  rcvd: 59

root@debiancla:~#
```

Como se aprecia en la imagen, en el primer caso simplemente se nos devuelve la dirección IP asociada con el host y su dirección IPv6. Por otro lado, dig devuelve los registros DNS relacionados.

Uso de netstat y ss

Cuando en un equipo se necesite ejecutar una serie de servicios de red es importante tener abiertos los puertos necesarios y solamente esos. Ese es uno de los puntos fundamentales para asegurar el sistema contra intrusiones externas.

Veamos cómo utilizar `ss` para ver todos los *sockets* TCP (combinaciones de nombre de equipo o dirección IP junto con puerto de escucha) abiertos dentro del equipo:

```
ss --all --tcp
```

El comando ss nos permite también inspeccionar conexiones por uno o varios puertos. Si deseamos ver las conexiones entrantes filtrando por uno o varios puertos TCP (digamos el 80 y el 22, por ejemplo), podemos utilizar la opción `sport` (de *source port*) de la siguiente manera:

```
ss -tn '( sport = :80 or sport = :22 )'
```

Si desde nuestro equipo se estuvieran estableciendo conexiones salientes a puertos específicos de un host remoto, podríamos utilizar `dport` (destination port) seguido de los puertos de destino.

```
root@debiancla:~# ss --all --tcp
State      Recv-Q Send-Q                          Local Address:Port
LISTEN     0      128                                         *:ssh
LISTEN     0      5                                           *:ipp
LISTEN     0      100                                         *:smtp
ESTAB      0      336                           192.168.0.170:ssh
LISTEN     0      128                                        :::http
LISTEN     0      128                                        :::ssh
LISTEN     0      100                                        :::smtp
root@debiancla:~# ss -tn '( sport = :80 or sport = :22 )'
State      Recv-Q Send-Q                            Local Address:Port
ESTAB      0      64                                192.168.0.170:22
root@debiancla:~# netstat -npltu
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Progr
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      738/sshd
tcp        0      0 0.0.0.0:631             0.0.0.0:*               LISTEN      1350/cups
tcp        0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      915/maste
tcp6       0      0 :::80                   :::*                    LISTEN      786/apach
tcp6       0      0 :::22                   :::*                    LISTEN      738/sshd
tcp6       0      0 :::25                   :::*                    LISTEN      915/maste
udp        0      0 0.0.0.0:631             0.0.0.0:*                           1352/cups
udp        0      0 0.0.0.0:59555           0.0.0.0:*                           675/avahi
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           675/avahi
udp6       0      0 :::5353                 :::*                                675/avahi
udp6       0      0 :::37739                :::*                                675/avahi
root@debiancla:~#
```

La imagen de arriba ilustra el uso de los comandos que acabamos de explicar, junto con el empleo de `netstat` para ver los puertos abiertos (tanto de TCP como UDP en el sistema). Si bien su uso ha quedado un tanto relegado con respecto a `ss` y a `ip`, todavía es común emplearlo para inspeccionar conexiones de red, tablas de enrutamiento (con `-r`) e interfaces (`-i`).