Network Commands

Most Posix systems will have the same set of commands you can call from the shell. Linux added the ip2 suite of tools, which provide the same functionality with some additional bells and whistles.

Classic Posix Command	**ip2 Equivalent**
`ifconfig`	`ip link`, `ip address`
`route`, `netstat -r`	`ip route`, `ip route get`

Most ip2 subcommands can be abbreviated, such as `ip addr` instead of `ip address`.

`ifconfig`

This command gives you information about all of the *network devices* on the host, whether configured or not. For example, on the course VM:

```
vmuser@f18marsh:~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe9c:c99f  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:9c:c9:9f  txqueuelen 1000  (Ethernet)
        RX packets 319  bytes 195592 (195.5 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 258  bytes 41861 (41.8 KB)
        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 1000  (Local Loopback)
        RX packets 46  bytes 3982 (3.9 KB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 46  bytes 3982 (3.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The first entry shows the `enp0s2` device. The `e` typically indicates that this is an ethernet device. The second entry shows the `lo` device, which is "loopback" or "local" (they're different names for the same thing).

Each one has a set of flags, in this case both devices are `UP` and `RUNNING`, which means they are ready to handle packets. The `mtu` is the maximum transmission unit for the connection to the next-hop through that device, which is the number of bytes a single packet can contain. 65536 is the largest size that an IP packet can be.

The address for the device is given by `inet` (or `inet6` for IPv6). The `netmask` defines the subnet size.

A device with a hardware address also has `ether` followed by that address.

The rest is mostly statistics for the device. `RX` means received data, and `TX` means transmitted data.

`ip link`

The ip2 suite separates link (device) information from address information. For the former, you use the `ip link` command (in the PDF version this will be cut off, as it will for some of the later commands):

```
vmuser@f18marsh:~$ ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP mode DEFAULT group defaul
    link/ether 08:00:27:9c:c9:9f brd ff:ff:ff:ff:ff:ff
```

Compare these with the information from `ifconfig`. The format is slightly different, and there's a little information that `ifconfig` didn't provide, like `qdisc`. This is a Linux-specific feature that the Posix command doesn't know about.

`ip address`

We can get the address information, as well. In fact, this provides us with essentially everything that `ifconfig` provides, just formatted differently (and with the Linux additions):

```
vmuser@f18marsh:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    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: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:9c:c9:9f brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute enp0s3
       valid_lft 72430sec preferred_lft 72430sec
    inet6 fe80::a00:27ff:fe9c:c99f/64 scope link
       valid_lft forever preferred_lft forever
```

Now we have the hardware type and address first, followed by the IP and IPv6 addresses. Note that the addresses are given in CIDR notation. The only thing we don't have here are the RX and TX statistics.

Routing Tables

The `route` command prints out the routing table for the host. Note that some systems, like MacOS (which is otherwise Posix), have a `route` command that behaves a bit differently. Again, from our VM:

```
vmuser@f18marsh:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         _gateway        0.0.0.0         UG    100    0        0 enp0s3
10.0.2.0        0.0.0.0         255.255.255.0   U     100    0        0 enp0s3
```

You can also use `netstat -r`, which should provide identical output.

The ip2 suite has `ip route` instead:

```
vmuser@f18marsh:~$ ip route
default via 10.0.2.2 dev enp0s3 proto dhcp metric 100
10.0.2.0/24 dev enp0s3 proto kernel scope link src 10.0.2.15 metric 100
```

You can see that the information is all there, in different format, with the notable addition of the source address the host will use when sending a new packet based on that forwarding rule.

You can also ask what rule a particular address will use:

```
vmuser@f18marsh:~$ ip route get 10.0.2.10
10.0.2.10 dev enp0s3 src 10.0.2.15 uid 1000
    cache
```

Connecting Layers 2 and 3

We can examine the ARP cache with either of the following:

```
vmuser@f18marsh:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
_gateway                 ether   52:54:00:12:35:02   C                     enp0s3
```

This is the standard Posix command. The fields should be fairly self-explanatory.

```
vmuser@f18marsh:~$ ip neighbor
10.0.2.2 dev enp0s3 lladdr 52:54:00:12:35:02 DELAY
```

This is the ip2 command (which can be abbreviated as short as `ip n`).

Obviously, on our VM there isn't much going on. Here's what we might see on our host:

```
16:44 ~ $ arp -an
? (10.104.80.1) at 0:0:c:7:ac:0 on en0 ifscope [ethernet]
? (172.26.4.1) at 2:e0:52:40:3f:40 on en8 ifscope [ethernet]
? (172.26.5.254) at cc:4e:24:d1:b0:0 on en8 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:0:0:fb on en8 ifscope permanent [ethernet]
```

This is on MacOS, so we have to provide `-a`. The `-n` prevents addresses from being converted to hostnames, as with other network commands. The format is also slightly different, as you can see.

Existing Network Connections

The existence of `netstat -r` might provide a hint that `netstat` can do other things, as well. Here's a particularly useful command:

```
vmuser@f18marsh:~$ netstat -taun
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
udp        0      0 0.0.0.0:42966           0.0.0.0:*
udp        0      0 127.0.0.53:53           0.0.0.0:*
udp        0      0 0.0.0.0:68              0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp6       0      0 :::50307                :::*
udp6       0      0 :::5353                 :::*
```

Here are what these flags mean:

Flag	Meaning
`-t`	match TCP sockets
`-u`	match UDP sockets
`-a`	show all, not just active, sockets
`-n`	just show numeric addresses/ports
`-p`	(not shown) the PID and process name (if permitted)

Note that on MacOS (and some other Posix platforms), `-u` tells `netstat` to display *Unix sockets*, which are for interprocess communication on the host.

Examining Connectivity

Our two workhorses here are `ping` and `traceroute`. Both take a number of options, but generally are called as

```
ping <destination>
```

Here's an example of `ping`:

```
vmuser@f18marsh:~$ ping -c 3 gizmonic.cs.umd.edu
PING gizmonic.cs.umd.edu (128.8.130.3) 56(84) bytes of data.
64 bytes from gizmonic.cs.umd.edu (128.8.130.3): icmp_seq=1 ttl=63 time=0.928 ms
64 bytes from gizmonic.cs.umd.edu (128.8.130.3): icmp_seq=2 ttl=63 time=1.57 ms
64 bytes from gizmonic.cs.umd.edu (128.8.130.3): icmp_seq=3 ttl=63 time=1.33 ms

--- gizmonic.cs.umd.edu ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.928/1.280/1.575/0.268 ms
```

This shows you we have a working network path to gizmonic, as well as the round-trip times. We lost no packets, and also get to see the statistics for the round-trip times.

`traceroute` lets us examine the path between us and a destination:

```
vmuser@f18marsh:~$ traceroute www.google.com
traceroute to www.google.com (216.58.217.100), 30 hops max, 60 byte packets
 1  _gateway (10.0.2.2)  0.240 ms  0.152 ms  0.084 ms
 2  router-604.cs.umd.edu (172.26.4.1)  2.840 ms  4.756 ms  6.638 ms
 3  csnat03.priv.cs.umd.edu (172.26.127.103)  0.791 ms  0.851 ms  1.129 ms
 4  128.8.127.190 (128.8.127.190)  9.209 ms  12.075 ms  15.525 ms
 5  * * *
 6  129.2.0.178 (129.2.0.178)  2.105 ms  1.412 ms  1.492 ms
 7  128.8.0.160 (128.8.0.160)  1.617 ms  1.676 ms  1.676 ms
 8  128.8.0.13 (128.8.0.13)  2.628 ms  2.510 ms  2.507 ms
 9  206.196.177.200 (206.196.177.200)  2.887 ms  2.982 ms  3.128 ms
10  mbp-t1-1720.maxgigapop.net (206.196.177.109)  4.431 ms  4.330 ms  4.147 ms
11  * * *
12  216.239.54.106 (216.239.54.106)  4.098 ms  4.033 ms 72.14.235.32 (72.14.235.32)  5.498 ms
13  iad23s42-in-f4.1e100.net (216.58.217.100)  4.172 ms 108.170.246.3 (108.170.246.3)  4.860 ms 108.170
```

You can suppress the conversion of IP addresses to names with the `-n` flag. The way `traceroute` works is that it sends a small packet with a very small TTL flag, and waits to see the ICMP Time Exceeded message, the sender of which is TTL hops away. Note that this is not reliable, as paths can vary from packet to packet.

Finding Other Hosts

Your simplest go-to for looking up a host's IP address given its name is to use the `host` command:

```
host gizmonic.cs.umd.edu
```

On older systems, you would use

```
nslookup gizmonic.cs.umd.edu
```

This has been deprecated on newer systems, so you should always use `host` when available. If you want more detail about the query and response, you can use `dig` instead:

```
dig gizmonic.cs.umd.edu
```

All of these have a number of options available to you to control what kinds of queries you perform. They are also able to perform *reverse lookups*. That is, given an IP address, they will tell you the (canonical) hostname:

```
host 128.8.130.3
dig -x 128.8.130.3
```

You can also find out who registered a domain with `whois`:

```
whois umd.edu
```

You can even find out who owns the subnet an address is in:

```
whois 128.8.130.3
```