Extracted from:

Small, Sharp, Software Tools

Harness the Combinatoric Power of Command-Line Tools and Utilities

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Small, Sharp, Software Tools

Harness the Combinatoric Power of Command-Line Tools and Utilities

Brian P. Hogan

edited by Tammy Coron

Small, Sharp, Software Tools

Harness the Combinatoric Power of Command-Line Tools and Utilities

Brian P. Hogan

The Pragmatic Bookshelf
Raleigh, North Carolina

ISBN-13: 978-1-68050-296-1
Book version: B1.0—November 7, 2018

Exploring Network Connections

Computers connect to other computers using an IP address and a port. For example, when you made a request using nc to google.com, you told it to connect on port 80, the default port for web traffic. And to complete that request, your computer needed to figure out which IP address google.com resolved to, so it made a request to a DNS server using port 53, the default port for DNS queries.

Ports allow multiple network connections from a single machine. Imagine that the IP address is the street address to an apartment complex, and each port is an apartment number in the building.

When you installed the openssh-server package, your Ubuntu machine started listening for incoming connections on port 22. When you connected to the server, your client made an outgoing connection on port 22.

Your OS makes all kinds of network connections to remote systems, and programs you install do as well. Unfortunately, so do malicious programs. It's not a bad idea to keep tabs on your computer's communication.

There are a handful of tools that let you see which ports are in use. The two you'll look at are netstat and ss.

netstat is older, and more universally-available on servers and Linux operating systems. Like ifconfig, it's also not supported anymore. You'll explore it first and then look at other options.

On your Ubuntu virtual machine, stop the SSH server:

```
(ubuntu) $ sudo systemctl stop sshd
```

Now, you'll use netstat to look at what's listening for incoming TCP connections. Execute this command on your Ubuntu virtual machine:

```
(ubuntu) $ netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 puzzles:domain          *:*                     LISTEN
tcp        0      0 localhost:ipp           *:*                     LISTEN
tcp6       0      0 ip6-localhost:ipp       [::]:*
```

The -l flag only displays servers or programs that are listening for connections. The -t flag only shows TCP connections.

Netstat shows the protocol, the number of bytes queued up for receiving and sending, the local address, the remote address, and the state. In this example, everything is normal. There are three entries listening for connections, but there's no data in either the receive queue or the send queue. The Foreign

Address field shows *:*, which indicates there's no remote connection, and the LISTEN state shows there there's no connection established yet.

If you're wondering what those things are, hold tight; you'll explore that in a bit. But first, start up the SSH server again:

```
(ubuntu) $ sudo systemctl start sshd
```

Then, look at the connections again:

```
(ubuntu) $ netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 puzzles:domain          *:*                     LISTEN
➤ tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 localhost:ipp           *:*                     LISTEN
➤ tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
```

This time you see two new entries in the output related to SSH.

Connect from your local machine to the SSH server:

```
(local)$ ssh brian@192.168.99.100
```

Then, on the Ubuntu virtual machine, look at the connections again, but this time use netstat -at. The -a switch looks at active connections as well as ones that are waiting:

```
(ubuntu) $ netstat -at
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 puzzles:domain          *:*                     LISTEN
tcp        0      0 *:ssh                   *:*                     LISTEN
tcp        0      0 localhost:ipp           *:*                     LISTEN
➤ tcp        0      0 192.168.99.100:ssh      192.168.99.1:61809      ESTABLISHED
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
tcp6       0      0 ip6-localhost:ipp       [::]:*                  LISTEN
```

You can see the connection between the machines now.

So what are those other entries in the list? On Linux systems like Ubuntu, you can see which program or process owns the connection by executing netstat with sudo and adding the -p switch. You'll need sudo to see information about ports lower than 1024:

```
$ sudo netstat -atp
Active Internet connections (servers and established)
Proto ... Local Address       Foreign Address     State       PID/Program name
tcp   ... puzzles:domain      *:*                 LISTEN      837/dnsmasq
tcp   ... *:ssh               *:*                 LISTEN      14317/sshd
tcp   ... localhost:ipp       *:*                 LISTEN      7024/cupsd
tcp   ... 192.168.99.100:ssh  192.168.99.1:61809  ESTABLISHED 14363/sshd: brian [
tcp6  ... [::]:ssh            [::]:*              LISTEN      14317/sshd
tcp6  ... ip6-localhost:ipp   [::]:*              LISTEN      7024/cupsd
```

This output shows that the dnsmasq and cups services are listening for connections. dnsmasq is a service for resolving hostnames, and cups is a service for printing. These are built-in services configured by default when you installed Ubuntu. But the output also shows which user is connected to the SSH server, which can be very helpful.

Unfortunately, not all versions of netstat support this option. For example, the BSD version on macOS won't show you this information. Thankfully there are workarounds that turn out to be a little better than netstat.

The lsof command lets you see which files are associated with processes. On a Linux-based system, everything is represented as a file, including network connections. This means you can use lsof to perform the same tasks that netstat performs.

To list all services listening for connections over TCP, execute this command:

```
(ubuntu) $ sudo lsof -nP -iTCP -sTCP:LISTEN
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dnsmasq   837 nobody    5u  IPv4  17380      0t0  TCP 127.0.1.1:53 (LISTEN)
cupsd    7024   root   10u  IPv6 118530      0t0  TCP [::1]:631 (LISTEN)
cupsd    7024   root   11u  IPv4 118531      0t0  TCP 127.0.0.1:631 (LISTEN)
sshd    15866   root    3u  IPv4 169492      0t0  TCP *:22 (LISTEN)
sshd    15866   root    4u  IPv6 169508      0t0  TCP *:22 (LISTEN)
```

The -n switch tells lsof not to resolve domain names, which makes it run a lot faster. The -iTCP' switch selects files associated with Internet addresses using the TCP protocol. The -sTCP:LISTEN selects only files in a listening state. From the results, you can see that the SSH server is running, as well as the dnsmasq and cups' services.

If you switch -sTCP:LISTEN with -sTCP:ESTABLISHED, you'll see active network connections:

```
(ubuntu) $ sudo lsof -nP -iTCP -sTCP:ESTABLISHED
COMMAND   PID  USER ... NODE NAME
sshd    15879  root ...  TCP 192.168.99.100:22->192.168.99.1:64220 (ESTABLISHED)
sshd    15905 brian ...  TCP 192.168.99.100:22->192.168.99.1:64220 (ESTABLISHED)
```

In this case, you see two listings for the open SSH connection. One represents the SSH server itself, running as root, and the other represents the established client connection.

Before you finish up, let's look at the ss command, which is the modern replacement for netstat. It's part of the iproute2 package on Linux systems.

To see listening TCP connections along with which user and process, execute ss with the -ltp switches:

```
(ubuntu) $ sudo ss -ltp
State   Recv-Q Send-Q  Local Address:Port    Peer Address:Port
LISTEN  0      5          127.0.1.1:domain            *:*
  users:(("dnsmasq",pid=837,fd=5))
LISTEN  0      128                *:ssh               *:*
  users:(("sshd",pid=15866,fd=3))
LISTEN  0      5          127.0.0.1:ipp               *:*
  users:(("cupsd",pid=7024,fd=11))
LISTEN  0      128               :::ssh              :::*
  users:(("sshd",pid=15866,fd=4))
LISTEN  0      5                ::1:ipp              :::*
  users:(("cupsd",pid=7024,fd=10))
```

The -l switch shows listening sockets, -t shows TCP only, and -p shows the associated process information.

Unfortunately, macOS doesn't support ss, so you'll have to stick with lsof.

These tools are essential for quickly identifying either open ports or ports that are already in use by a development server. They're also helpful to identify which ports you need to open in your firewall.

Let's look at another versatile tool you should get to know when working with networks.

Using netcat

The netcat program, or nc, is the "Swiss Army Knife" of networking tools. With this one tool, you can connect to remote systems, transfer files, and even scan ports.

You can use nc to determine if certain services are running by scanning the ports of those services. For example, you can scan to see if the web server is running by scanning for port 80, the default port for web servers:

```
$ nc -z -v www.example.com 80
```

If there's a web server running, you'll see this:

```
Connection to www.example.com 80 port [tcp/http] succeeded!
```

You can also scan ranges of ports. For example, to scan for all ports from 22 (SSH) to 80 (Web), execute this command:

```
$ nc -z -v www.example.com 20-80
```

Making web requests

You already used cURL to grab web pages, but netcat can do that too. However, netcat makes you do it a little more interactively.

First, type this command:

```
$ nc www.google.com 80
```

You'll be greeted by a blank line; netcat is expecting some input. You're going to craft your own HTTP request. Type the following:

```
GET / HTTP/1.1
```

Then, press the `ENTER` key twice and you'll see the source code for the Google home page stream out to your screen.

You can add more data to the request. For example, when you send a request to a web server, the browser identifies itself, and often time sends along the URL of the page the request came from, also known as the `referer` (which is actually spelled incorrectly, believe it or not.) You can use `nc` to specify those headers, or even make them up.

Try it out. Make a new request:

```
$ nc www.google.com 80
```

Then, type the following lines in, pressing `ENTER` after each line:

```
GET / HTTP/1.1
Host: google.com
User-Agent: Internet Explorer
Referrer: awesomeco.com
```

Press the `ENTER` key twice to send the request.

This makes a request with your own crafted request headers, which let you pretend to use Internet Explorer for the request. Why would we do this? Sometimes web developers write code to prevent people from using certain browsers, so you can use the `User-Agent` header to pretend to be something you're not and bypass these kinds of restrictions. Of course, a more legitimate usage is to correctly identify the program you're using.

Serving files

You can use netcat to serve files if you combine it with a little bit of shell scripting:

```
$ cat << EOF > hello.txt
This is a text file served from netcat
EOF
```

Now, execute this command to make netcat listen for connections on port 8080 and serve the hello.txt file:

```
$ while true; do nc -l 8000 < hello.txt; done
```

This loops indefinitely, listening for connections on port 8000, and then reads in the file, sending its contents to anything that connects. In another terminal, make a request with curl:

```
$ curl localhost:8000
This is a text file served from netcat
```

Return to the original terminal and press Ctrl-c to stop the loop.

You can use this approach to serve a web pages. Create a web page named index.html with some text:

```
$ cat << EOF > index.html
<h1>Hi from netcat</h1>
EOF
```

To make a browser render the HTML instead of just displaying the source, you'll have to craft a response the browser understands. Instead of just reading in a file, create an HTTP response. Send the text HTTP/1.1 200 OK, followed by two blank lines, followed by the contents of the file:

```
$ while true; \
do echo -e "HTTP/1.1 200 OK\n\n $(cat index.html)" | \
nc -l 8000; done
```

With this running, fire up a browser and go to http://localhost:8000 to see your page.

This is just one more example of how diverse netcat is. But you're not quite done.

Realtime Chat with netcat

You can use nc as an improvised chat system. This isn't entirely useful, but it's a fun exercise to explore, as it shows how netcat can send data in real time.

On one computer, type:

The code tag should not be here.

This starts a chat server listening on port 1337. You can then connect to this server using another machine with nc, specifying the IP address of the chat server:

The code tag should not be here.

At this point, you can type messages on either machine, and the other machine will display them. Pressing CTRL-C breaks the connection for both machines.

You can use netcat for lots of other things too. You can use it to send files or create secure internet connections. You've just scratched the surface of this tool. Its primary use is for ad-hoc network diagnostics, but it really is a networking multitool.

Security conscious folks should know that netcat does everything in an unsecured manner. Use this only on trusted networks.

Your Turn

These additional exercises will help you get more comfortable with the tools you used in this chapter.

1. Using the tools you learned about in this chapter, answer the following questions:
2. Who is the administrative contact for the wordpress.com domain?
3. Which domain will need to be renewed first; heroku.com or google.com?
4. How many IP addresses are associated with heroku.com?
5. Who has more IP addresses associated with their domain: Facebook, Google, Twitter, Wikipedia, or Amazon?
6. Which of the following IP addresses belongs to a Comcast cable subscriber? Which one of these belongs to Google?
   - 4.2.2.1
   - 137.28.1.17
   - 24.23.51.253
   - 45.23.51.32
   - 8.8.8.8
7. Classify these IP addresses into Class A, B, and C:
   - 1.1.1.1
   - 137.28.1.17
   - 24.23.51.253
   - 45.23.51.32

- 192.167.23.1
- 8.8.8.8

8. Use `traceroute` on a few of your favorite web sites. What similarities do you see between each? What differences do you see?

9. Use `cURL` to inspect the headers sent by the following sites.

   - http://twitter.com
   - https://pragprog.com
   - http://news.ycombinator.com
   - https://reddit.com
   - http://automattic.com

   If any sites redirect to a new site, use `curl` to make an additional request using the `location` header.

10. Use `nc` to make the same connections.

11. Use cURL with the Open Weather API[4] to find the weather forecast for your area. You'll need to register for an API key before you can access the API. What command did you end up using?

12. Explain the difference between these two commands:

    - `scp -r data username@host:/data`
    - `scp -r data/* username@host:/data`

    When would you use one over the other?

13. Identify all of the established connections on your local machine.

What You Learned

The tools you used in this chapter will become an essential part of your arsenal. You'll revisit a few of them later when you work with networks. You may need these tools to diagnose networking issues of your own, work with APIs, or transfer data between computers on your network.

Next, you'll take the commands and concepts you've learned so far and use them to create scripts of commands you can run over and over to automate tasks.

4. https://openweathermap.org/api