How to open a TCP/UDP socket in a bash shell

Last updated on October 21, 2020 by Dan Nanni

Suppose you want to open a TCP/UDP socket on a Linux server for various reasons. For example, you want to check if a specific address/port is reachable. Or you want to fetch a remote web page or invoke a restful API for testing. Or you want to connect to a remote IRC server, etc. However, what if the Linux server you are on is very restrictive? On that server, none of standard tools such as `netcat`, `curl` or `wget` may be available, and you are pretty much left with the `bash` shell only.

In fact, one of built-in features of `bash` shell is to open TCP/UDP sockets via `/dev/tcp` (and `/dev/udp`) device file. In the rest of this tutorial, let's find out **how to open a TCP/UDP socket, and read to and write from the socket in bash shell**.

Open or Close a TCP/UDP Socket in Bash Shell

In a nutshell, you can open a TCP/UDP socket using the following syntax in `bash` shell.

```
$ exec {file-descriptor}
<>/dev/{protocol}/{host}/{port}
```

The `file descriptor` is a unique non-negative integer associated with each socket. File descriptors 0, 1 and 2 are reserved for `stdin`, `stdout` and `stderr`, respectively. Thus you must specify `3` or higher (whichever is unused) as a file descriptor.

`<>` implies that the socket is open for both reading and writing. Depending on your need, you can open a socket for read-only (`<`) or write-only (`>`).

The `protocol` field can be either tcp or udp. The `host` and `port` fields are self-explanatory.

For example, to open a bi-directional TCP socket for `xmodulo.com` with `HTTP` port and file descriptor `3`:

```
$ exec 3<>/dev/tcp/xmodulo.com/80
```

Once opened, a read/write socket can be closed using the following syntax. The first command close an input connection, while the latter closes an output connection.

```
$ exec {file-descriptor}<&-
$ exec {file-descriptor}>&-
```

Read from or Write to a TCP/UDP Socket in Bash Shell

Once a socket is opened, you can write a message to or read a message from the socket.

To write a message stored in `$MESSSAGE` to a socket:

```
$ echo -ne $MESSAGE >&3
$ printf $MESSAGE >&3
```

To read a message from a socket and store it in `$MESSAGE`:

```
$ read -r -u -n $MESSAGE <&3
$ MESSAGE=$(dd bs=$NUM_BYTES count=$COUNT <&3 2>
/dev/null)
```

TCP/UDP Socket Examples in Bash Shell

Here I present several shell script examples that open and use a TCP socket.

1. Fetch a remote web page and print its content.

```
#!/bin/bash
exec 3<>/dev/tcp/xmodulo.com/80
echo -e "GET / HTTP/1.1rnhost:
xmodulo.comrnConnection: closernrn" >&3
cat <&3
```

2. Display a remote SSH server version.

```
#!/bin/bash
exec 3</dev/tcp/192.168.0.10/22
timeout 1 cat <&3
```

In fact, the above script can be shortened to the following one-liner:

```
#!/bin/bash
timeout 1 cat </dev/tcp/192.168.0.10/22
```

3. Display the current time from nist.gov.

```
#!/bin/bash
cat </dev/tcp/time.nist.gov/13
```

4. Check the Internet connectivity.

```
#!/bin/bash

HOST=www.mit.edu
PORT=80

(echo >/dev/tcp/${HOST}/${PORT}) &>/dev/null
if [ $? -eq 0 ]; then
    echo "Connection successful"
else
    echo "Connection unsuccessful"
fi
```

5. Perform TCP port scanning against a remote host.

```
#!/bin/bash
host=$1
port_first=1
port_last=65535
for ((port=$port_first; port<=$port_last;
port++))
do
  (echo >/dev/tcp/$host/$port) >/dev/null 2>&1 &&
echo "$port open"
done
```

Final Notes

Opening a socket in `bash` requires that the `bash` shell have *net-redirections* enabled (i.e., compiled with `--enable-net-redirections`). Old distributions may have this feature disabled for `bash`, in which case you will encounter the following error.

```
/dev/tcp/xmodulo.com/80: No such file or
directory
```

Besides `bash`, socket support is known to be available in other shells such as `ksh` or `zsh`.