Creating a Simple TCP Socket Server in Bash

Last updated: March 2, 2024

Written by: baeldung (https://www.baeldung.com/linux/author/baeldung)

Networking (https://www.baeldung.com/linux/category/networking)

bash (https://www.baeldung.com/linux/tag/bash) **netcat (https://www.baeldung.com/linux/tag/netcat)**

socat (https://www.baeldung.com/linux/tag/socat)

1. Overview

Creating a TCP (/cs/udp-vs-tcp#tcp) server application listening on a certain port is trivial when we use a high-level programming language. However, we might want to have a quick and simple socket (/cs/port-vs-socket#what-is-a-socket) server without learning a new programming language. Moreover, we might be unable to install or use the compiler or the interpreter on the system due to constrained privileges or resources.

In this tutorial, we'll use the tools available in Bash to create a simple TCP socket server.

All commands and examples in this guide have been tested on Debian 12 running GNU Bash 5.2.15, *netcat-openbsd* 1.219, *socat* 1.7.4.4, and *ncat* 7.93.

2. Using *netcat*

To begin, we'll use *netcat* (/linux/netcat-command). *netcat* is a flexible networking utility that we can use for a variety of purposes involving TCP (/cs/udp-vs-tcp#tcp) and UDP (/cs/udp-vs-tcp#udp) protocols.

2.1. Installation

netcat is commonly available on most Unix-like systems. However, **there are usually two packages of *netcat* already automatically installed on our system**:

```
$ apt search netcat
...
netcat-openbsd/stable,now 1.219-1 amd64 [installed,automatic]
TCP/IP swiss army knife

netcat-traditional/stable,now 1.10-47 amd64 [installed]
TCP/IP swiss army knife
...
```

netcat-traditional (https://packages.debian.org/bookworm/netcat-traditional) is the original *netcat* codebase. On the other hand, ***netcat-openbsd* (https://packages.debian.org/sid/netcat-openbsd) is more secure and feature-rich compared to the traditional version**. *netcat-openbsd* derives from the

OpenBSD project (https://man.openbsd.org/OpenBSD-6.6/nc.1#AUTHORS).

In case they're not installed on our system, **we can install the *netcat-openbsd* package with the *apt* (/linux/yum-and-apt#2-installation-of-a-package) command**:

```
$ sudo apt install netcat-openbsd
$ nc -h
OpenBSD netcat (Debian patchlevel 1.219-1)
...
```

After the installation finishes, we can run the *nc* binary with the *-h* option to check its version.

Both *netcat- packages contain the same binary – *nc***. However, **on Debian 12, the default binary is the *nc* binary from the *netcat-openbsd* package**:

```
$ whereis nc
nc: /usr/bin/nc /usr/share/man/man1/nc.1.gz
$ readlink -f /usr/bin/nc
/usr/bin/nc.openbsd
```

Referring to the commands above, we used the *whereis* (/linux/which-command-problems#3-thewhereis-command) command to learn the location of the *nc* binary (/linux/absolute-directory-of-file). Then, we used the *readlink* (/linux/absolute-directory-of-file#1-readlink) command to determine whether the path is an absolute path or a symbolic link (/linux/symbolic-and-hard-links#symbolic-links) (symlink) and retrieve the target of the symlink.

2.2. Creating a TCP Socket Server

Let's use *netcat* to create a TCP socket server listening on port *8080*:

```
$ nc -k -l 8080
```

The *-k* option tells *netcat* to continue listening after the first connection completes rather than exiting the program. The *nc* command processes the incoming connections sequentially, not concurrently. In addition, the *-l* option instructs *netcat* to operate in listening mode.

While the server's running, we open another terminal as a client to connect to the server by using the same command – *nc*:

```
$ nc localhost 8080
```

Once the client is connected to the server, we can type random characters to send to the server, followed by pressing the *Enter* key. Subsequently, those characters should appear on the server-side terminal.

Furthermore, the socket connection is a two-way communication, meaning the server can also send messages to the client:

00:00 00:22

We can close the socket connection by simply pressing *Ctrl + C* on either the client- or server-side terminal.

We've successfully created a simple sequential TCP socket server using *netcat* in Bash.

3. Using *socat*

socat (/linux/socat-command) (SOcket CAT) is a feature-rich networking utility. It's designed as a multi-purpose relay utility and offers many features, such as port forwarding and redirection (/linux/socat-command#3-forwarding-a-stream), proxying/tunneling (/linux/create-tunnel-local-ports#using-the-socat-utilities), and virtual terminal/serial port (/linux/make-virtual-serial-port#create-a-virtual-serial-port-using-socat-utility).

However, we can also it to create a simple TCP socket server.

3.1. Installation

The *socat* package is also available on most Unix-like systems under the canonical name *socat*:

```
$ apt search socat
socat/stable,now 1.7.4.4-2 amd64
multipurpose relay for bidirectional data transfer
$ sudo apt install socat
$ socat -V
...
socat version 1.7.4.4 on 06 Nov 2022 08:15:51 ...
...
```

Once the installation is finished, we can check its version by passing the *-V* option.

3.2. Creating a TCP Socket Server

Let's create a TCP socket server listening on port 8080:

```
$ socat TCP-LISTEN:8080,fork -
```

Let's review the options:

- *TCP-LISTEN:8080*: listen on port *8080*
- *fork*: fork a new process for each incoming connection
- *–*: use the standard input and output for communication with the connected clients

By having the *fork* option, the TCP socket server is capable of handling concurrent requests (/cs/web-server-concurrent-requests-one-port#handling-concurrent-requests-on-a-single-port). However, since each child process inherits the standard input/output descriptors (/linux/tcp-ip-connections-limit#file-descriptors) from its parent process, this can lead to unexpected behavior when we try to send messages from the server.

Once the server is running, let's open another terminal as the client and create a socket connection to the server:

```
$ socat STDIO TCP4:localhost:8080
```

We also used the *socat* command to open the socket connection.

Let's break down the command:

- *STDIO*: use the standard input/output (*stdin*/*stdout*) for communication
- *TCP4:localhost:8080*: connect to the *localhost* server on port *8080*

Once the client is connected, we can simply type the message on the terminal, followed by pressing the *Enter* key. As a result, the message should appear on the server-side terminal.

Additionally, the socket connection enables two-way communication, so we can send and receive messages from both client- and server-side terminals:

00:00 00:37

If we want to close the socket connection, we can simply press *Ctrl + C* on either the client- or server-side terminal.

We've managed to create a simple concurrent TCP socket server using *socat* in a Bash environment.

4. Using *ncat*

ncat (https://nmap.org/ncat/) is a reimplementation or *netcat*. However, it's designed with security in mind and offers broader features compared to *netcat*.

4.1. Installation

The *ncat* package is available on most Unix-like systems under the name *ncat*:

```
$ apt search ncat
...
ncat/stable 7.93+dfsg1-1 amd64
NMAP netcat reimplementation
...
$ sudo apt install ncat
$ ncat --version
Ncat: Version 7.93 ( https://nmap.org/ncat )
```

After installing the package, we can verify the installation by checking its version, as shown in the output above.

4.2. Creating a TCP Socket Server

Since *ncat* is a reimplementation of *netcat*, they both share similar options.

Let's create a TCP socket server:

```
$ ncat -k -l 8080
```

The *-k* option means the *ncat* process keeps running after the client closes the connection. In addition, the *-l* option tells *ncat* to run in listening mode. Similar to *netcat*, **the *ncat* command processes the incoming connections sequentially, not concurrently**.

While the server's running, we open another terminal to act as a client to connect to the server by using the same command – *ncat*:

```
$ ncat localhost 8080
```

After the client connects to the server, we can send messages to the server by typing in the terminal and pressing the *Enter* key. Subsequently, those messages should appear on the server-side terminal.

Since the socket connection is a two-way communication, the server can also send messages to the client:

00:00 00:26

To close the socket connection, we can simply press *Ctrl + C* on either the client- or server-side terminal.

At this point, **we've also successfully created a simple sequential TCP socket server using *ncat* in a Bash environment**.

5. Conclusion

In this article, we learned how to create a simple TCP socket server in a Bash environment. We explored three networking utilities that are commonly available in most Linux official repositories: *netcat*, *socat*, and *ncat*. They can all be utilized to create a socket server, as well as to act as a socket client.

While all three are networking utilities, there are some differences. The *netcat* command is a simple networking utility to do basic networking tasks, while *ncat* is a reimplementation of the *netcat* codebase with security in mind. Therefore, both tools have many similar command options. On the other hand, ***socat* is a more advanced version of the other two and offers even more features**.

Additionally, both *netcat* and *ncat* process incoming connections sequentially, whereas *socat* supports concurrency (/cs/web-server-concurrent-requests-one-port#concurrent-requests), allowing it to handle multiple connections simultaneously (/cs/web-server-concurrent-requests-one-port#handling-concurrent-requests-on-a-single-port).

Comments are open for 30 days after publishing a post. For any issues past this date, use the Contact form on the site.

CATEGORIES

ADMINISTRATION (/LINUX/CATEGORY/ADMINISTRATION)
FILES (/LINUX/CATEGORY/FILES)
FILESYSTEMS (/LINUX/CATEGORY/FILESYSTEMS)
INSTALLATION (/LINUX/CATEGORY/INSTALLATION)
NETWORKING (/LINUX/CATEGORY/NETWORKING)
PROCESSES (/LINUX/CATEGORY/PROCESSES)
SCRIPTING (/LINUX/CATEGORY/SCRIPTING)
SEARCH (/LINUX/CATEGORY/SEARCH)
SECURITY (/LINUX/CATEGORY/SECURITY)
WEB (/LINUX/CATEGORY/WEB)

SERIES

LINUX ADMINISTRATION (/LINUX/LINUX-ADMINISTRATION-SERIES)
LINUX FILES (/LINUX/LINUX-FILES-SERIES)
LINUX PROCESSES (/LINUX/LINUX-PROCESSES-GUIDE)
LINUX SECURITY TUTORIALS (HTTPS://WWW.BAELDUNG.COM/LINUX/LINUX-SECURITY-SERIES)