What is / dev / null in Linux?

Technically, '/ dev / null' is a virtual device file. For related programs, they are treated as real files.

Technically, **'/ dev / null'** is a virtual device file. For related programs, they are treated as real files. Utilities may require data from this source type and the operating system will provide data for them. But, instead of reading from the drive, the operating system will create this data flexibly. An example of such a file is **'/ dev / zero'.**

However, in this case, you will write to a device file. Anything you write to '/ dev / null', is removed and forgotten. To understand why this is useful, you must first have a basic understanding of standard output (standard output) and standard error in Linux or * nix operating systems.

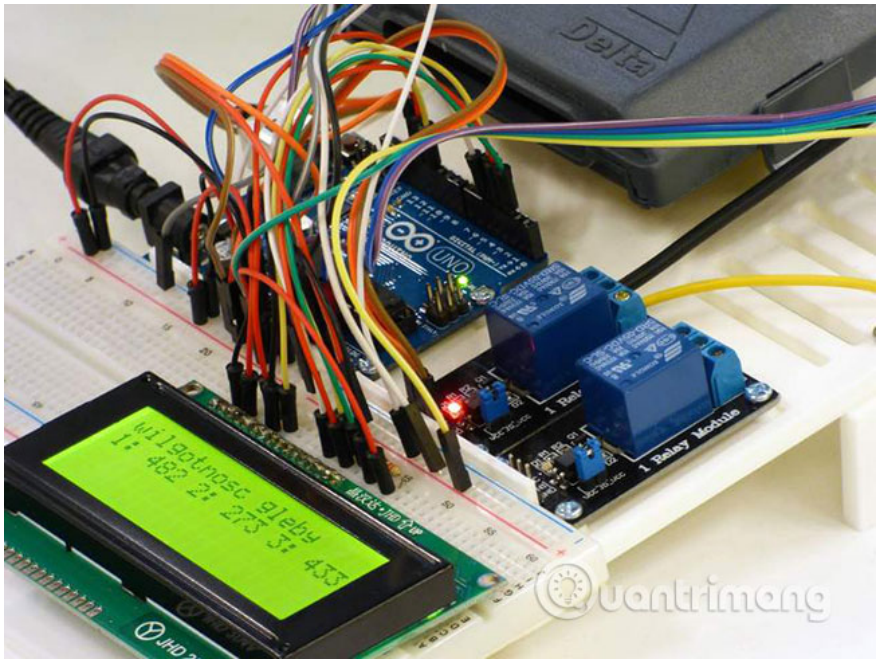
Learn about / dev / null in Linux

1. stdout and stderr
2. Use / dev / null to get rid of the output you don't need
3. Redirect all output to / dev / null
4. Other examples may be useful when redirecting to / dev / null

stdout and stderr

A command line utility can create two types of output. Standard output is sent to **stdout.** Error sent to **stderr.**

By default, stdout and stderr are linked to the terminal window (or console). This means everything is sent to stdout and stderr is usually displayed on the screen. But through shell navigation, you can change this behavior. For example, you can redirect stdout to a file. This way, instead of displaying the output on the screen, the data will be saved to a file for you to read later - or you can redirect the stdout to the physical device, for example, the LED screen or the LCD technology. number.



TipsMake.com has a complete article about Pipe in Unix / Linux, if you want to learn more.

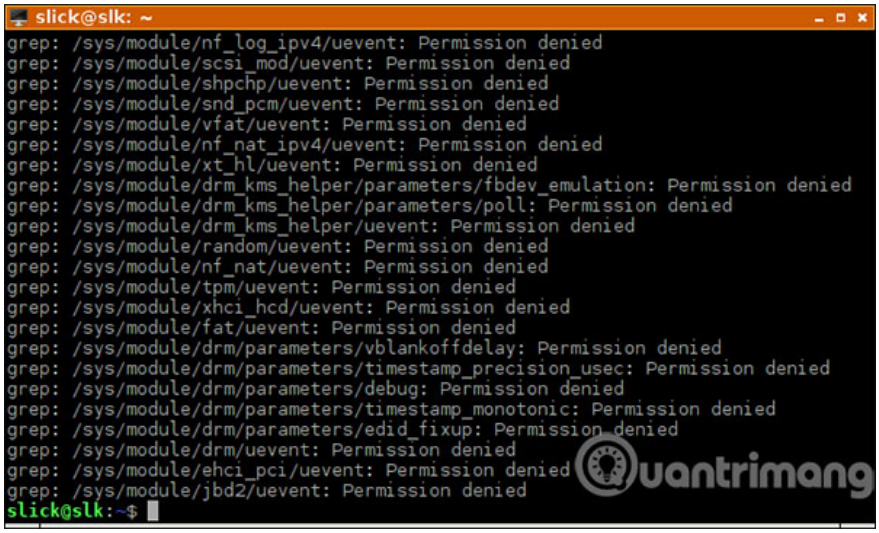
1. With **2**> you redirect standard error messages. For example: `2>/dev/null` or `2>/home/user/error.log` .
2. With **1**> you redirect the standard output.
3. With **&**> you redirect both standard errors and standard outputs.

Use / dev / null to get rid of the output you don't need

As mentioned above, there are two types of outputs: Standard output and standard error. The first use case is to filter out the output or standard error. It is easier to understand when considering through a practical example. For example, you are looking for a string in **'/ sys'** to find files related to source settings.

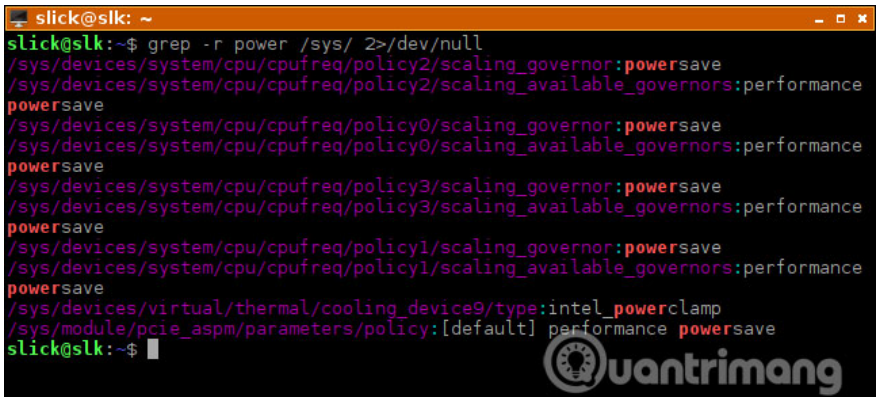
```
grep -r power /sys/
```

There will be many files that non-root users cannot read. This results in many **'Permission denied'** errors **.**



These things clutter the output and make it hard for you to find the results you're looking for. Because the '**Permission denied**' errors are part of stderr, you can redirect them to '/ dev / null'.

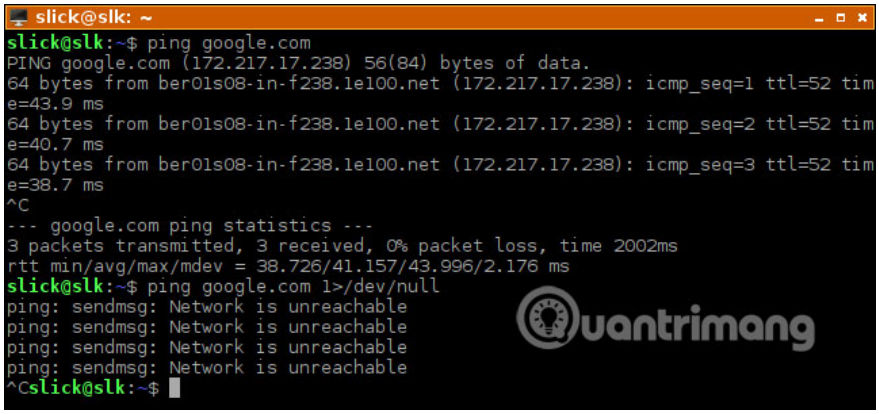
```
grep -r power /sys/ 2>/dev/null
```



As you can see, the results of this command are much easier to read.

In other cases, doing the opposite (filtering out standard output so that you can only see errors) can be helpful.

```
ping google.com 1>/dev/null
```



The screenshot above shows that, without redirection, ping will display its normal output when possible to the destination machine. In the second command, nothing is displayed while the network is online, but as soon as it is disconnected, only the error message is displayed.

You can redirect both stdout and stderr to two different locations.

```
ping google.com 1>/dev/null 2>error.log
```

In this case, the stdout messages will not be displayed completely and the error message will be saved to the **error.log** file **.**

Redirect all output to / dev / null

Sometimes, removing all output can be very helpful. There are two ways to do this.

```
grep -r power /sys/ >/dev/null 2>&1
```

String > / **dev** / **null** means sending stdout to / dev / null, and the second, **2> & 1,** means sending stderr to stdout. In this case, you must treat stdout as **& 1,** instead of simply **1** . Recording **2> 1** will only redirect stdout to a file named **1.**

The important thing to note here is the very important order. If you reverse the navigation parameters like this:

```
grep -r power /sys/ 2>&1 >/dev/null
```

The command will not work as intended. The reason is because as soon as **2> & 1** is interpreted, stderr is sent to stdout and displayed on the screen. Next, stdout is removed when sent to '/ dev / null'. The end result is that you will see errors on the screen instead of eliminating all output. If you can't remember the correct order, there is a simpler, easier way to navigate:

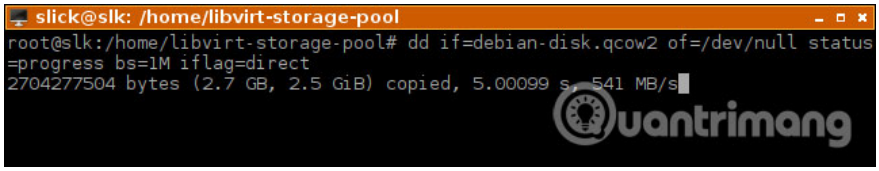
```
grep -r power /sys/ &>/dev/null
```

In this case, **&>** / **dev** / **null** is equivalent to requiring "redirecting both stdout and stderr to this position".

Other examples may be useful when redirecting to / dev / null

Suppose you want to see how fast your drive can read serial data. The test is not extremely accurate, but the results are also reliable enough. You can use **dd** for this, but **dd** can export stdout or be instructed to write to the file. With **of** = / **dev** / **null,** you can ask **dd to** write to this virtual file. Even, you don't need to use shell redirection here. **if** = specifies the location of the input file to be read; **of** = specifies the name of the output file, where the data is written.

```
dd if=debian-disk.qcow2 of=/dev/null status=progress bs=1M iflag=direct
```



In some cases, you may want to consider how quickly you can download from the server. But you will not want to write to the drive if not needed. Simply put, don't write to a regular file, write '/ dev / null'.

```
wget -O /dev/null http://ftp.halifax.rwth-aachen.de/ubuntu-releases/18.04/ubuntu
```

Hopefully, the examples in this article can inspire you to find your own creative ways to use '/ dev / null'.

If you know of an interesting use case for this special device file, leave a comment in the comment section below!

Hope you are succesful.