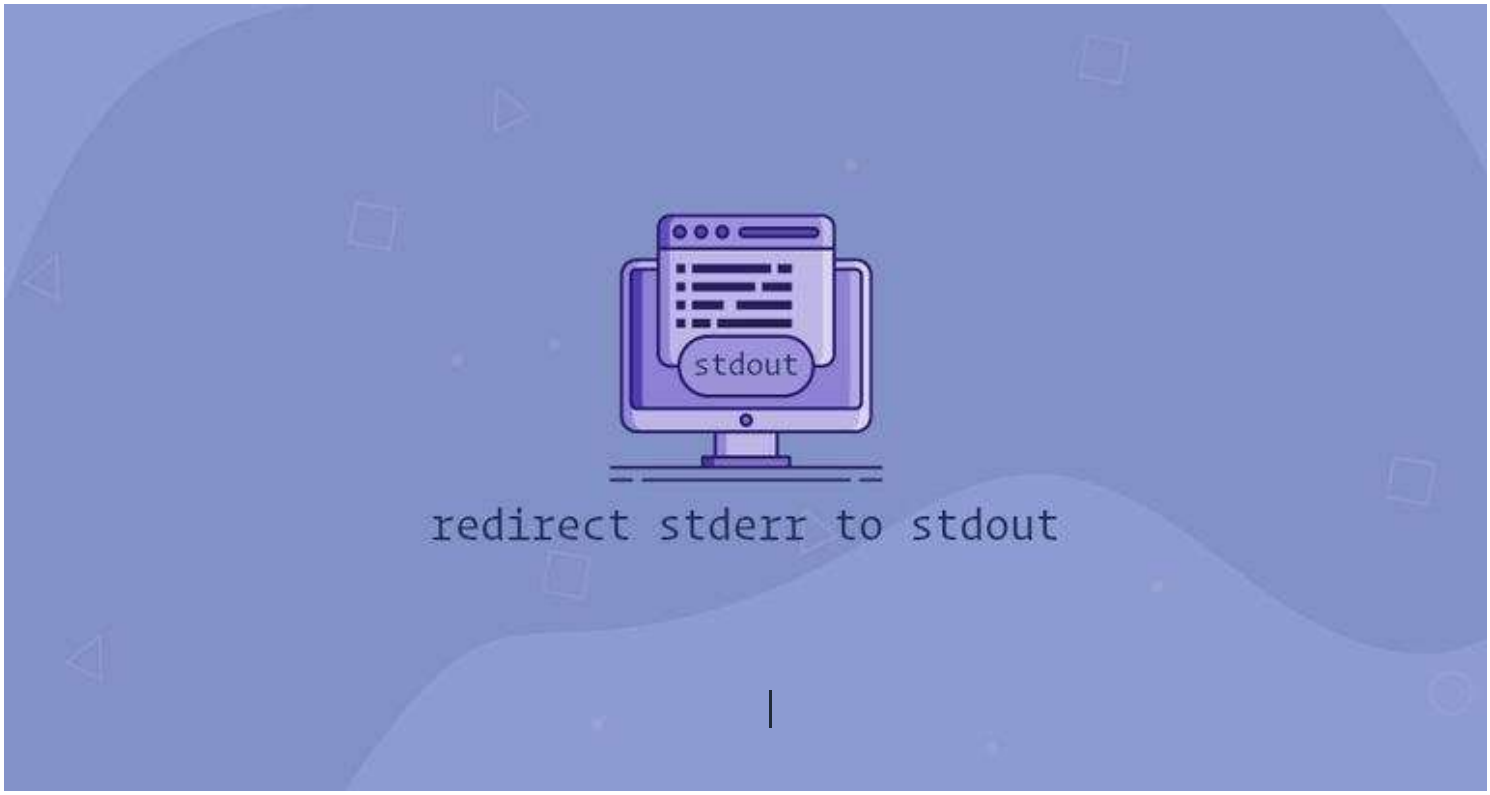
How to Redirect stderr to stdout in Bash

Posted Jun 5, 2020 • 3 min read



When redirecting the output of a command to a file or piping it to another command, you might notice that the error messages are printed on the screen.

In Bash and other Linux shells, when a program is executed, it uses three standard I/O streams. Each stream is represented by a numeric file descriptor:

- `0 - stdin` , the standard input stream.
- `1 - stdout` , the standard output stream.
- `2 - stderr` , the standard error stream.

A file descriptor is just a number representing an open file.

The input stream provides information to the program, generally by typing in the keyboard.

Redirecting Output

Redirection is a way to capture the output from a program and send it as input to another program or file.

Streams can be redirected using the `n>` operator, where `n` is the file descriptor number.

When `n` is omitted, it defaults to `1`, the standard output stream. For example, the following two commands are the same; both will redirect the command output (`stdout`) to the file.

```
$ command > file
```

```
$ command 1> file
```

To redirect the standard error (`stderr`) use the `2>` operator:

```
$ command 2> file
```

You can write both `stderr` and `stdout` to two separate files:

```
$ command 2> error.txt 1> output.txt
```

To suppress the error messages from being displayed on the screen, redirect `stderr` to `/dev/null`:

Redirecting `stderr` to `stdout`

When saving the program's output to a file, it is quite common to redirect `stderr` to `stdout` so that you can have everything in a single file.

To redirect `stderr` to `stdout` and have error messages sent to the same file as standard output, use the following:

```
$ command > file 2>&1
```

`> file` redirect the `stdout` to `file`, and `2>&1` redirect the `stderr` to the current location of `stdout`.

The order of redirection is important. For example, the following example redirects only `stdout` to `file`. This happens because the `stderr` is redirected to `stdout` before the `stdout` was redirected to `file`.

```
$ command 2>&1 > file
```

```
$ command &> file
```

Conclusion

Understanding the concept of redirections and file descriptors is very important when working on the command line.

To redirect `stderr` and `stdout`, use the `2>&1` or `&>` constructs.

If you have any questions or feedback, feel free to leave a comment.

bash terminal

Related Articles

JUN 8, 2020

Bash Exit Command and Exit Codes

MAY 31, 2020

Bash printf Command

Write a comment