Understanding Shell Script's idiom: 2>&1

brianstorti.com/understanding-shell-script-idiom-redirect

November 10, 2015

When we are working with a programming or scripting language, we are constantly using some idioms, some things that are done in *this certain way*, the common solution to a problem. With Shell Script this is not different, and a quite common idiom, but not so well understood, is the `2>&1`, like in
`ls foo > /dev/null 2>&1`.
Let me explain what is going on here and why this works the way it does.

A quick introduction to I/O redirection

Simply put, redirection is the mechanism used to send the output of a command to another place. For instance, if we just `cat` a file, its output will be printed in the screen, by default:

```
$ cat foo.txt
foo
bar
baz
```

But we can redirect this output to another place. Here, for example, we are redirecting it to a file called `output.txt`:

```
$ cat foo.txt > output.txt

$ cat output.txt
foo
bar
baz
```

Note that in the first `cat`, we don't see any output in the screen. We changed the **standard output** (`stdout`) location to a file, so it doesn't use the screen anymore.

It's also important to know that there are this other place, called **standard error** (`stderr`), to where programs can send their error messages. So if we try to `cat` a file that doesn't exist, like this:

```
$ cat nop.txt > output.txt
cat: nop.txt: No such file or directory
```

Even if we redirect the `stdout` to a file, we still see the error output in the screen, because we are redirecting just the standard output, not the standard error.

And a quick introduction to file descriptors

A file descriptor is nothing more that a positive integer that represents an open file. If you have 100 open files, you will have 100 file descriptors for them.

The only caveat is that, in Unix systems, *everything is a file*. But that's not really important now, we just need to know that there are file descriptors for the Standard Output (`stdout`) and Standard Error (`stderr`).

In plain English, it means that there are "ids" that identify these two locations, and it will always be `1` for `stdout` and `2` for `stderr`.

Putting the pieces together

Going back to our first example, when we redirected the output of `cat foo.txt` to `output.txt`, we could rewrite the command like this:

```
$ cat foo.txt 1> output.txt
```

This `1` is just the file descriptor for `stdout`. The syntax for redirecting is `[FILE_DESCRIPTOR]>`, leaving the file descriptor out is just a shortcut to `1>`.

So, to redirect `stderr`, it should be just a matter of adding the right file descriptor in place:

```
# Using stderr file descriptor (2) to redirect the errors to a file
$ cat nop.txt 2> error.txt

$ cat error.txt
cat: nop.txt: No such file or directory
```

At this point you probably already know what the `2>&1` idiom is doing, but let's make it official.

You use `&1` to reference the value of the file descriptor 1 (`stdout`). So when you use `2>&1` you are basically saying "Redirect the `stderr` to the same place we are redirecting the `stdout`". And that's why we can do something like this to redirect both `stdout` and `stderr` to the same place:

```
$ cat foo.txt > output.txt 2>&1

$ cat output.txt
foo
bar
baz

$ cat nop.txt > output.txt 2>&1

$ cat output.txt
cat: nop.txt: No such file or directory
```

Recap

- There are two places programs send output to: Standard output (`stdout`) and Standard Error (`stderr`);

- You can redirect these outputs to a different place (like a file);
- File descriptors are used to identify `stdout` (1) and `stderr` (2);
- `command > output` is just a shortcut for `command 1> output`;
- You can use `&[FILE_DESCRIPTOR]` to reference a file descriptor value;
- Using `2>&1` will redirect `stderr` to whatever value is set to `stdout` (and `1>&2` will do the opposite).

And if you want to learn more about Shell Script, I highly recommend the Classic Shell Scripting book.