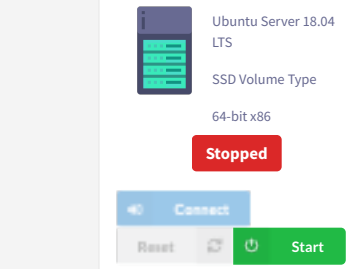
Bash and Linux - Redirections

30 minutes Normal



DataScientest • com

Introduction to Linux

Redirections

Standard output, Standard Error and Standard Input

We have seen in the previous part that when a command is run and prints something, it is written in the **Standard output**, meaning the console. The symbols > and >> allow to redirect the standard output to a file. The console itself is called the **Standard input** where you can write your commands. The third component is the **Standard Error** which is where errors are printed. By default, the standard error is redirected to the standard output. However, it is possible to separate these two.

Run this command to generate an error

```
cat no_file
```

We can use 2> or 2>> to redirect the standard error in a file:

Run this command to print the error into `errors_file`

```
cat no_file 2> errors_file

cat errors_file
```

The standard input, output and error have each their **file descriptor**:

- 0 for the standard input
- 1 for the standard output
- 2 for the standard error

> is actually an alias for 1>. We can use < or 0< to redirect content to a command. For example:

```
head < root_content.txt
```

This is very useful for some programs that require interactions.

Create a file named `greetings.py` containing the following lines

```
name = input("State your name:\n")
print("Hi {} !".format(name))
```

Execute it by using `python3 greetings.py`

Create a file named `first_name.txt` with `Daniel` inside

Show / Hide solution

Run the following command

```
python3 greetings.py < first_name.txt
```

We can also use redirections to separate errors from outputs by chaining them:

Run the following command

```
ca 1>fic 2>log
cat fic
cat log
```

To print both errors and outputs in the same file, we can use `2>&1` at the end of the command.

```
ls -l > fic 2>&1
```

Pipe operator

Sometimes, we need to chain commands and provide the output of one command to another command. To do so, we can use `|`.

Run the following command

```
ls / | grep bin
```

The first part of the command is a simple display of the content of the `/` folder. Then we take this output and pass it to the `grep` function. `grep` is a very popular tool to filter text. Here, it is going to print only the lines that contain `bin`.

Display all the commands that you ran so far containing `file`

To do so, we can use `history`

Show / Hide solution

&& operator

The `&&` operator is going to be used to perform multiple commands in a specific order but without any link between them:

Run this command

```
ls -l && python3 --version && mkdir Test
```

Note that if one of the command fails, the following ones will not be run.

Validated