Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
○ Linux Introduction

Data Streams

Contents

Overview

Linux Data Streams can be used to pass information along different processes using redirection and is also a part of how a terminal work fundamentally using Standard Input (`stdin`), Standard Output (`stdout`) and Standard Error (`stderr`).

Stream Types

Standard Output (`stdout`)

Every system using Linux will have a default place for output to go, for all applications.
Your shell (bash, sh, zsh, etc.) is watching this location for changes.
So when you run a command that's why the terminal updates, displaying information that's useful such as errors or some indication that your command was successful.

The location for Standard Output is `/dev/stdout`.

Standard Input (`stdin`)

This is a default location to listen for information.
Anything sent to this file can be read by an application or script.
Usually what gets sent to here is characters typed from your keyboard, for your terminal to use.

Running this example in a terminal will listen for changes to the `/dev/stdin` file.
As you type something and then hit enter, it will be outputted back to you.

```
while read line; do
    echo ${line}
done < /dev/stdin
```

Standard Error (`stderr`)

Very similar to Standard Output and Input however this is the location for errors.
Having errors as a seprate stream means that they can be managed on their own, this might be for printing it out as red text or directing it to another location such as an error log file.

Redirection

Redirect To A File (`>`)

The `>` character can be used to redirect `stdout` to a given file.
Note that `stderr` by default is not redirected to the file and is just printed on to the terminal as usual.

Here the standard output for the `ls -al /etc` command is redirected to a file in `/tmp/test.txt`.

```
ls -al /etc > /tmp/test.txt
```

This is an example of a command that fails, if you run it you will notice that the error is still outputted to the terminal:

```
ls -al /doesnt/exist > /tmp/test.txt
```

Appending (`>>`)

By using `>` the contents of the provided file will be overwritten, this behaviour can be changed to append to the file by using `>>` instead:

```
ls -al /etc >> /tmp/test.txt
```

Standard Input (`<`)

Standard input can be redirected from a file by using `<`.

A common use case for this is running SQL scripts against the `mysql` client:

```
mysql < script.sql
```

Heredocs (`<<`)

Here Documents is a type of special-purpose code block.

A block of data can be sent as an input stream to a command, here's a simple example of using a here document with the cat command:

```
cat << EOF
Line 1
Line 2
Line 3
EOF
```

`EOF` in the example above is used as a `delimiting identifier`.
This declares when the input stream starts and stops, so the enclosed information is directed to `stdin`.

Here's an example for using a here document with SSH, multiple commands are being sent to the SSH command at once:

```
ssh user@host << EOF
echo 1
echo 2
echo 3
EOF
```

Piping

Pipes (`|`) can be used to redirect `stdout` to another process, converting into `stdin`.

One use case for using a pipe could be running a remote script.
The `curl` command returns the script from a URL:

```
# script to install docker
curl https://get.docker.com
```

The output of the `curl` command (the script itself) can be redirected as `stdin` to `bash`, which will execute the commands in the script:

```
# don't try running this unless you don't mind having docker installed on your machine...
curl https://get.docker.com | sudo bash
```

Redirection with File Descriptors

We can use file descriptors as a way of capturing data streams for standard input, output and error.

POSIX (Portable Operating System Interface) a set of standards which defines how these can be accessed, amongst many other standards for compatibility between operating systems.

According to POSIX here:

- STDIN_FILENO
  Standard input value, stdin. Its value is 0.
- STDOUT_FILENO
  Standard output value, stdout. Its value is 1.
- STDERR_FILENO
  Standard error value, stderr. Its value is 2.

So what this is saying is, Standard Input can be accessed with `0`, Standard Output with `1` and Standard Error with `2`.

Standard Error Redirection

To a File

The file descriptor for Standard Error is `2`, so we can reference this when redirecting output to a file.
See below for the errors of a command that is failing being redirected to an error log file:

```
ls -al /I/will/not/work > /tmp/error.log
```

We'll be able to see that the Standard Output still just goes to the terminal for commands that work fine:

```
ls -al / > /tmp/error.log
```

To Standard Output

There are times when we will want to combine errors and normal output together into the same file.
To do this we can combine the error and output streams by redirecting the error stream to standard output using `2>&1`.

Here's an example of two commands, the first one is successful and the second one fails.
Both the standard output and error streams will be combined and sent to a log file:

```
(ls -al / && ls -al sdfgst43qg4wergs) > /tmp/log.txt 2>&1
```

Tutorial

Simulating Running a Remote Script

Prerequisites

You will need to clone down this repository using Git and change into that directory.

```
git clone https://gitlab.com/qacdevops/data-streams-remote-script.git
cd data-streams-remote-script
```

Now run the deploy.sh script to run the application.

```
. ./deploy.sh
```

Now, you must **leave this terminal running** and open up a new terminal.

> Note, if you are working on a virtual machine, you will need to SSH back to that machine on this new terminal.

View the Remote Script

We can see the script by using curl.

```
curl localhost:5000/script
```

or going to the same URL or your external IP address in your browser. Please note, if you are using a virtual machine, you will need port 5000 to be open.

Execute the Remote Script

We can now execute this remote script by piping it to bash:

```
curl localhost:5000/script | bash
```

This should print the expected message to your terminal.

```
Hello from remote script
```

Exercises

SSH Commands

Use a Here Document to send multiple commands to a remote machine via SSH.