Redirection and pipes

Max Carter-Brown
Anglia Ruskin University, Wellcome Sanger Institute
max.carter-brown@aru.ac.uk

Contents

1. Outcomes

By the end of this document, you should be able to:

1. Understand the difference between standard input (STDIN), standard output (STDOUT), and standard error (STDERR).
2. Understand what pipes are and how they can be used to chain commands together.
3. Understand how to redirect the output of a command to a file.

2. Introduction

There are three default *files* (from 0-2).

1. **STDIN** (standard input = 0) is the file where the operating system sends data to the command.
2. **STDOUT** (standard output = 1) is the file where the operating system sends the command's output.
3. **STDERR** (standard error = 2) is the file where the command sends its error messages (in general).

3. The three *files*

STDOUT is usually the screen - the command sends its output to the screen. In a contrived example:

Shell 1: Printing "Hello world!" to STDOUT

```
echo "Hello, world!"
```

STDERR is also usually the screen - the command sends its error messages to the screen. In another contrived example:

Shell 2: Printing "ls: /nonexistent: No such file or directory" to STDERR

```
ls /nonexistent
```

Such usage of STDERR is usually accompanied by an error code (in this case, the error code is 1). This can be useful for scripting, as you can check the error code to see if the command was successful or not.

STDIN is usually the keyboard - you are sending characters from the keyboard to the terminal or command. STDIN can also be a file or another command's output - we will see this later when using pipes.

4. Pipes

A pipe is a way to connect the STDOUT of one command to the STDIN of another command. This allows you to chain commands together.

Shell 3: Counting the number of lines, words, and bytes in "Hello, world!"

```
echo "Hello, world!" | wc
```

In this example "Hello world!" would have been printed to the screen. However, the pipe we have added means that what would have printed to the screen (STDOUT) is instead sent to the command `wc` as STDIN (which counts the number of lines/words/bytes in the input). The result of `wc` is then printed to the screen (STDOUT).

Using pipes, we can chain commands together to perform more complex operations. For example, we could list the files in the current directory and count the number of files:

Shell 4: Counting the number of files and directories in the current directory

```
ls | wc -l
```

In bioinformatics and other fields, pipes are used extensively to chain together commands to perform complex operations on data. We will explore some simple to more complex examples of this later in the course.

5. Redirection

Redirection is the process of changing where the output of a command goes. You can redirect the output of a command to a file, or to another command. We've actually seen an example of this already - when we used the pipe `|` to redirect the output of one command to another command. Another common use of redirection is to redirect the output to a file (i.e. saving to a file).

Shell 5: Redirecting "Hello, world!" to a file called "hello.txt"

```
echo "Hello, world!" > hello.txt
```

This is great for saving files - and note that if the file already exists, it will be overwritten. If you want to append to an existing file, you can use `>>`.

Shell 6: Appending "Hello, again!" to the file "hello.txt"

```
echo "Hello, again!" >> hello.txt
```

We can check the status of our redirection using the `cat` command. `cat` will print the contents of a file to STDOUT, which we can then use as STDIN for the next command.

Shell 7: Showing there are two lines in the file "hello.txt"

```
cat hello.txt | wc -l
```

Questions

1. Create your own simple pipeline using `ls -l` and `head` to list the first file in the current directory.
2. Is there an easy way to print to STDERR using the `echo` command? If not, why not?
3. What would the following command do: `echo "Hello, world!" 2> error.txt`? What happened? What does the file contain? Why?