Stderr and Stdout

Data Streams

Streams are a way of transferring data from one point to another.

The 3 standard streams in *nix are `stdin`, `stdout`, and `stderr`.

- `stdin` takes data from the shell to the program.
- `stdout` and `stderr` take data from the program to the shell.
    - They are functionally equivalent

Streams can be treated the same as files in many simple cases.

Usage in C

```
fprintf(file, format, ...)
```

In C, writing to a file is functionally the same as writing to a built-in stream. To do so, you can pass `stdout` or `stderr` to `printf`.

Throwing an error by outputting to `stderr` won't stop the program, so you'll have to stop the program manually if that's what you need.

C Code:

```c
fprintf(stdout, "%s", "stdout gets piped!\n");
fprintf(stderr, "%s", "stderr doesn't get piped\n");
fprintf(stdout, "%s", "stdout gets piped again!\n");
```

Output:

```
stdout gets piped!
stderr doesn't get piped
stdout gets piped again!
```

Redirects

```
$ ./prog [redirect] output.txt
```

- `>`: Stdout of program is put into file, `1>` works too
- `2>`: Stderr of program is put into file
- `&>`: Stdout and stderr is put into file
- `<`: Stdin is put into file, `0>` works too

Can be chained:

```
$ ./ prog > output.txt 2> errors.txt
```

Pipes

```
$ ./prog | grep -n "pipe"
```

Output:

```
stderr doesn't get piped
1:stdout gets piped!
2:stdout gets piped again!
```

Notice: The final output gets misordered since the first and third output lines get piped into grep and resolved with the grep command while the second output line gets resolved with the program.