IO Redirection and Pipes

IO Redirection and Pipes

One of the more powerful features of the shell is the ability to connect commands in various ways. For example, you can take the output of one command and use it as the input for another command. Or you can have any command take its input from a file instead of the keyboard. This is accomplished through Input-Output (I/O) redirection and pipes.

Standard Input, Output and Error

Every command run by the shell (including the shell itself) has a concept of Standard Input, Standard Output, and Standard Error (STDIN, STDOUT and STDERR, respectively). STDIN is where the command usually sends its input, and by default this will be the keyboard. STDOUT is where the command sends its output, and by default this will be the terminal in which it is launched. STDERR is where the command usually prints any error messages, usually also the terminal in which it is launched.

Not all commands make use of all of these facilities. Some commands (e.g. **ls**) don't take any input. They take arguments and options, but don't accept any input. Other commands (e.g. **zwrite**) require input (i.e. the body of the zephyrgram) in order to function correctly.

It's also important to note that just because a command generates a file as output does not mean that its STDOUT is a file, and just because a command can accept a file as input does not mean that its STDIN is a file. Some commands do read from and write to files, but I/O redirection (discussed in the next section) can apply to **any** shell command, not just those which are specifically designed to deal with files.

Redirecting Input and Output

Perhaps the most common use case is redirecting a command's standard output to a file. Obviously this is most important for commands which generate a lot of output, but for simplicity we will be using the **fortune** command, which generates a random quote or witticism:

```
joeuser@athena$ fortune
Be different: conform.
joeuser@athena$
```

To redirect a command's output (STDOUT) to a file, use the greater-than sign (**>**) followed by the name of the file. For example:

```
joeuser@athena$ fortune > myfortune.txt
joeuser@athena$
```

Note that nothing appeared to happen, since the output was all sent to the file myfortune.txt. If you view the file using the **cat** command (described in detail later), you will see your fortune (if you're following along, the output will be different because a random fortune is chosen each time):

```
joeuser@athena$ cat myfortune.txt
Never be led astray onto the path of virtue.
joeuser@athena$
```

When you redirect STDOUT to a file, any existing file of the same name will be overwritten. If you wish to add output to a file, you can use the append operation, which is 2 greater-than signs (**>>**). So, to add yet another fortune to your file:

```
joeuser@athena$ fortune >> myfortune.txt
joeuser@athena$
```

Again, no output was printed on the screen. But if we view the file again, we'll see two fortunes:

```
joeuser@athena$ cat myfortune.txt
Never be led astray onto the path of virtue.
Tomorrow, you can be anywhere.
joeuser@athena$
```

Now we'll look at redirecting a command's standard input using the less-than sign (**<**). As noted earlier, the **zwrite** command (to send a zephyrgram) uses STDIN to get the contents of the message (you type it, and end with a '.' on a line by itself). This means we can tell zwrite to get its STDIN from a file instead of the keyboard. We can use the file created in the previous example:

```
joeuser@athena$ zwrite joeuser < myfortune.txt
Message queued for joueuser... sent
joeuser@athena$
```

Note that zwrite did not prompt you to type a message – instead it used the contents of the file as the body of the zephyr. You should have

received a zephyr with two fortunes.

Standard Error

You'll note that we haven't yet addressed Standard Error (STDERR). As noted above, STDERR is usually the same place as STDOUT. However, when we redirect STDOUT to file, STDERR will still be displayed on the screen. If we try to redirect the fortune command to a file, but include an option that the fortune command doesn't support, we'll see an error message, and the file we create will be empty:

```
joeuser@athena$ fortune -q > myfortune.txt
fortune: invalid option – 'q'
fortune-mod version 9708
fortune [-afilosw] [-m pattern] [-n number] [ [#%] file/directory/all]
joeuser@athena$ cat myfortune.txt
joeuser@athena$
```

The file is empty because the fortune command did not print anything to its STDOUT, and instead displayed an error on STDERR. The error did not end up in the file because only STDOUT was redirected to the file, not STDERR. It is possible to redirect STDOUT and STDERR individually or together, though that is not covered in this document.

Pipes

Pipes (the pipe character **|** is usually located on the key directly below the Backspace key – it is sometimes printed on the keyboard as two vertical bars to distinguish it from the letter I) are used to take the standard output of one command and immediately send it to the standard input of another command. Using pipes, we can shorten the previous examples to a single command, which takes the output of the **fortune** command and sends it to the **zwrite** command:

```
joeuser@athena$ fortune | zwrite joeuser
Message queued for joeuser... sent
joeuser@athena$
```