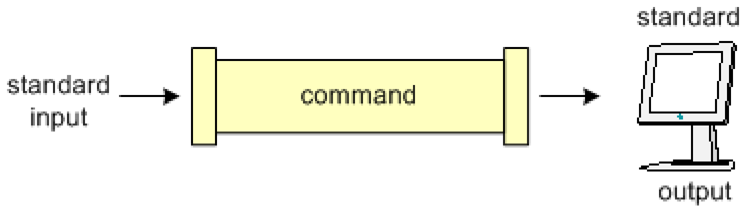
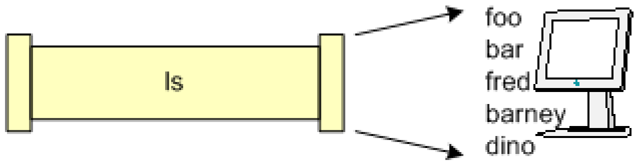
I/O Redirection and Pipes

In typical Unix installations, commands are entered at the keyboard and output resulting from these commands is displayed on the computer screen. Thus, input (by default) comes from the terminal and the resulting output (stream) is displayed on (or directed to) the monitor. Commands typically get their input from a source referred to as **standard input** (stdin) and typically display their output to a destination referred to as **standard output** (stdout) as pictured below:



As depicted in the diagram above, input flows (by default) as a stream of bytes from standard input along a channel, is then manipulated (or generated) by the command, and command output is then directed to the standard output. The ls command can then be described as follows; there is really no input (other than the command itself) and the ls command produces output which flows to the destination of stdout (the terminal screen), as below:



The notations of standard input and standard output are actually implemented in Unix as files (as are most things) and referenced by integer file descriptors (or channel descriptors). The file descriptor for standard input is 0 (zero) and the file descriptor for standard output is 1. These are not seen in ordinary use since these are the default values.

Input/Output Redirection

Unix provides the capability to change where standard input comes from, or where output goes using a concept called Input/Output (I/O) redirection. I/O redirection is accomplished using a redirection operator which allows the user to specify the input or output data be redirected to (or from) a **file**. Note that redirection always results in the data stream going to or coming from a file (the terminal is also considered a file).

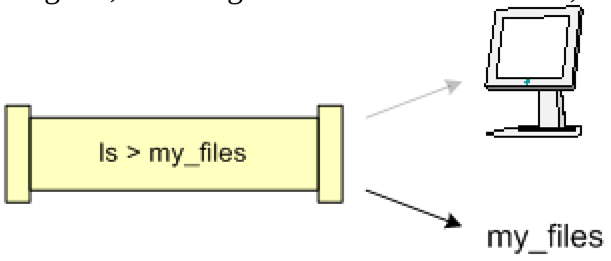
The simplest case to demonstrate this is basic **output redirection**. The output redirection operator is the > (greater than) symbol, and the general syntax looks as follows:

```
        command > output_file_spec
```

Spaces around the redirection operator are not mandatory, but do add readability to the command. Thus in our ls example from above, we can observe the following use of output redirection:

```
        $ ls > my_files [Enter]
        $
```

Notice there is no output appearing after the command, only the return of the prompt. Why is this, you ask? This is because all output from this command was *redirected* to the file my_files. Observe in the following diagram, no data goes to the terminal screen, but to the file instead.



Examining the file as follows results in the contents of the my_files being displayed:

```
        $ cat my_files [Enter]
        foo
        bar
        fred
        barney
        dino
        $
```

In this example, if the file my_files does not exist, the redirection operator causes its creation, and if it does exist, the contents are overwritten. Consider the example below:

```
        $ echo "Hello World!" > my_files [Enter]
        $ cat my_files [Enter]
        Hello World!
```

Notice here that the previous contents of the my_files file are gone, and are replaced with the string "Hello World!" Note also that when using redirection, the output file is created first, then the command left of the redirection operator is executed. Observe the following:

```
        $ cat my_files [Enter]
        Hello World!
        $ cat my_files > my_files [Enter]
        $ cat my_files [Enter]
        $
```

Often we wish to add data to an existing file, so the shell provides us with the capability to append output to files. The append operator is the >>. Thus we can do the following:

```
        $ ls > my_files [Enter]
        $ echo "Hello World!" >> my_files [Enter]
        $ cat my_files [Enter]
        foo
        bar
        fred
        barney
        dino
        Hello World!
```

The first output redirection creates the file if it does not exist, or overwrites its contents if it does, and the second redirection appends the string "Hello World!" to the end of the file. When using the append redirection operator, if the file does not exist, >> will cause its creation and append the output (to the empty file).

The ability also exists to redirect the standard input using the **input redirection** operator, the < (less than) symbol. Note the point of the operator implies the direction. The general syntax of input redirection looks as follows:

```
        command < input_file_spec
```

Looking in more detail at this, we will use the **wc** (**w**ord **c**ount) command. The wc command counts the number of lines, words and bytes in a file. Thus if we do the following using the file created above, we see:

```
        $ wc my_files [Enter]
               6       7      39   my_files
```

where the output indicates 6 lines, 7 words and 39 bytes, followed by the name of the file wc opened.

We can also use wc in conjunction with input redirection as follows:

```
        $ wc < my_files [Enter]
               6       7      39
```

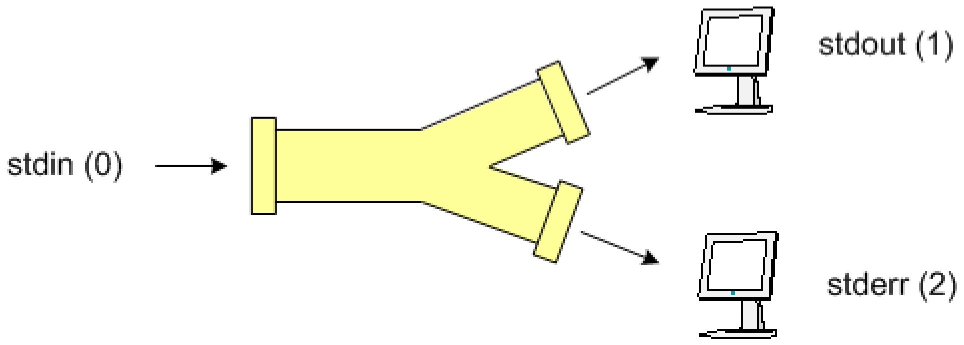
Note here that the numeric values are as in the example above, but with input redirection, the file name is not listed. This is because the wc command does not know the name of the file, only that it received a stream of bytes to count.

Someone will certainly ask if input redirection and output redirection can be combined, and the answer is most definitely yes. They can be combined as follows:

```
        $ wc < my_files > wc_output [Enter]
        $
```

There is no output sent to the terminal screen since all output was sent to the file wc_output. If we then looked at the contents of wc_output, it would contain the same data as above.

To this point, we have discussed the standard input stream (descriptor 0) and the standard output stream (descriptor 1). There is another output stream called standard error (stderr) which has file descriptor 2. Typically when programs return errors, they return these using the standard error channel. Both stdout and stderr direct output to the terminal by default, so distinguishing between the two may be difficult. However each of these output channels can be redirected independently. Refer to the diagram below:



The standard error redirection operator is similar to the stdout redirection operator and is the 2> (two followed by the greater than, with no spaces) symbol, and the general syntax looks as follows:

Syntax :

```
command_1 | command_2 | command_3 | .... | command_N
```

Example :

1. Listing all files and directories and give it as input to more command.

```
$ ls -l | more
```