UNIX - SHELL INPUT/OUTPUT REDIRECTIONS

Most Unix system commands take input from your terminal and send the resulting output back to your terminal. A command normally reads its input from a place called standard input, which happens to be your terminal by default. Similarly, a command normally writes its output to standard output, which is also your terminal by default.

Output Redirection:

The output from a command normally intended for standard output can be easily diverted to a file instead. This capability is known as output redirection:

If the notation > file is appended to any command that normally writes its output to standard output, the output of that command will be written to file instead of your terminal:

Check following **who** command which would redirect complete output of the command in users file.

```
$ who > users
```

Notice that no output appears at the terminal. This is because the output has been redirected from the default standard output device (the terminal) into the specified file. If you would check *users* file then it would have complete content:

```
$ cat users
oko         tty01   Sep 12 07:30
ai          tty15   Sep 12 13:32
ruth        tty21   Sep 12 10:10
pat         tty24   Sep 12 13:07
steve       tty25   Sep 12 13:03
$
```

If a command has its output redirected to a file and the file already contains some data, that data will be lost. Consider this example:

```
$ echo line 1 > users
$ cat users
line 1
$
```

You can use >> operator to append the output in an existing file as follows:

```
$ echo line 2 >> users
$ cat users
line 1
line 2
$
```

Input Redirection:

Just as the output of a command can be redirected to a file, so can the input of a command be redirected from a file. As the greater-than character > is used for output redirection, the less-than character < is used to redirect the input of a command.

The commands that normally take their input from standard input can have their input redirected from a file in this manner. For example, to count the number of lines in the file *users* generated above, you can execute the command as follows:

```
$ wc -l users
2 users
$
```

Here it produces output 2 lines. You can count the number of lines in the file by redirecting the standard input of the wc command from the file *users*:

```
$ wc -l < users
2
$
```

Note that there is a difference in the output produced by the two forms of the wc command. In the first case, the name of the file users is listed with the line count; in the second case, it is not.

In the first case, wc knows that it is reading its input from the file users. In the second case, it only knows that it is reading its input from standard input so it does not display file name.

Here Document:

A *here document* is used to redirect input into an interactive shell script or program.

We can run an interactive program within a shell script without user action by supplying the required input for the interactive program, or interactive shell script.

The general form for a here document is:

```
command << delimiter
document
delimiter
```

Here the shell interprets the << operator as an instruction to read input until it finds a line containing the specified delimiter. All the input lines up to the line containing the delimiter are then fed into the standard input of the command.

The delimiter tells the shell that the here document has completed. Without it, the shell continues to read input forever. The delimiter must be a single word that does not contain spaces or tabs.

Following is the input to the command **wc -l** to count total number of line:

```
$wc -l << EOF
 This is a simple lookup program
 for good (and bad) restaurants
 in Cape Town.
EOF
3
$
```

You can use *here document* to print multiple lines using your script as follows:

```
#!/bin/sh

cat << EOF
This is a simple lookup program
for good (and bad) restaurants
in Cape Town.
EOF
```

This would produce following result:

```
This is a simple lookup program
for good (and bad) restaurants
in Cape Town.
```

The following script runs a session with the vi text editor and save the input in the file test.txt.

```
#!/bin/sh

filename=test.txt
vi $filename <<EndOfCommands
i
This file was created automatically from
a shell script
^[
ZZ
EndOfCommands
```

If you run this script with vim acting as vi, then you will likely see output like the following:

```
$ sh test.sh
Vim: Warning: Input is not from a terminal
$
```

After running the script, you should see the following added to the file test.txt:

```
$ cat test.txt
This file was created automatically from
a shell script
$
```

Discard the output:

Sometimes you will need to execute a command, but you don't want the output displayed to the screen. In such cases you can discard the output by redirecting it to the file /dev/null:

```
$ command > /dev/null
```

Here command is the name of the command you want to execute. The file /dev/null is a special file that automatically discards all its input.

To discard both output of a command and its error output, use standard redirection to redirect STDERR to STDOUT:

```
$ command > /dev/null 2>&1
```

Here 2 represents STDERR and 1 represents STDOUT. You can display a message on to STDERR by redirecting STDIN into STDERR as follows:

```
$ echo message 1>&2
```

Redirection Commands:

Following is the complete list of commands which you can use for redirection:

Command	**Description**
pgm > file	Output of pgm is redirected to file
pgm < file	Program pgm reads its input from file.
pgm >> file	Output of pgm is appended to file.

n > file	Output from stream with descriptor n redirected to file.
n >> file	Output from stream with descriptor n appended to file.
n >& m	Merge output from stream n with stream m.
n <& m	Merge input from stream n with stream m.
<< tag	Standard input comes from here through next tag at start of line.
|	Takes output from one program, or process, and sends it to another.

Note that file descriptor 0 is normally standard input (STDIN), 1 is standard output (STDOUT), and 2 is standard error output (STDERR).