

I/O Redirection

In this lesson, we will explore a powerful feature used by many command line programs called *input/output redirection*. As we have seen, many commands such as `ls` print their output on the display. This does not have to be the case, however. By using some special notation we can *redirect* the output of many commands to files, devices, and even to the input of other commands.

Standard Output

Most command line programs that display their results do so by sending their results to a facility called *standard output*. By default, standard output directs its contents to the display. To redirect standard output to a file, the ">" character is used like this:

```
[me@linuxbox me]$ ls > file_list.txt
```

In this example, the `ls` command is executed and the results are written in a file named `file_list.txt`. Since the output of `ls` was redirected to the file, no results appear on the display.

Each time the command above is repeated, `file_list.txt` is overwritten (from the beginning) with the output of the command `ls`. If you want the new results to be *appended* to the file instead, use ">>" like this:

```
[me@linuxbox me]$ ls >> file_list.txt
```

When the results are appended, the new results are added to the end of the file, thus making the file longer each time the command is repeated. If the file does not exist when you attempt to append the redirected output, the file will be created.

Standard Input

Many commands can accept input from a facility called *standard input*. By default, standard input gets its contents from the keyboard, but like standard output, it can be redirected. To redirect standard input from a file instead of the keyboard, the "<" character is used like this:

```
[me@linuxbox me]$ sort < file_list.txt
```

In the above example we used the [sort](#) command to process the contents of `file_list.txt`. The results are output on the display since the standard output is not redirected in this example. We could redirect standard output to another file like this:

```
[me@linuxbox me]$ sort < file_list.txt > sorted_file_list.txt
```

As you can see, a command can have both its input and output redirected. Be aware that the order of the redirection does not matter. The only requirement is that the redirection operators (the "<" and ">") must appear after the other options and arguments in the command.

Pipes

By far, the most useful and powerful thing you can do with I/O redirection is to connect multiple commands together with what are called *pipes*. With pipes, the standard output of one command is fed into the standard input of another. Here is my absolute favorite:

```
[me@linuxbox me]$ ls -l | less
```

In this example, the output of the `ls` command is fed into `less`. By using this "`| less`" trick, you can make any command have scrolling output. I use this technique all the time.

By connecting commands together, you can accomplish amazing feats. Here are some examples you'll want to try:

Examples of commands used together with pipes

<i>Command</i>	<i>What it does</i>
<code>ls -lt head</code>	Displays the 10 newest files in the current directory.
<code>du sort -nr</code>	Displays a list of directories and how much space they consume, sorted from the largest to the smallest.
<code>find . -type f -print wc -l</code>	Displays the total number of files in the current working directory and all of its

subdirectories.

Filters

One class of programs you can use with pipes is called *filters*. Filters take standard input and perform an operation upon it and send the results to standard output. In this way, they can be used to process information in powerful ways. Here are some of the common programs that can act as filters:

Common filter commands

Program	What it does
<u>sort</u>	Sorts standard input then outputs the sorted result on standard output.
<u>uniq</u>	Given a sorted stream of data from standard input, it removes duplicate lines of data (i.e., it makes sure that every line is unique).
<u>grep</u>	Examines each line of data it receives from standard input and outputs every line that contains a specified pattern of characters.
<u>fmt</u>	Reads text from standard input, then outputs formatted text on standard output.
<u>pr</u>	Takes text input from standard input and splits the data into pages with page breaks, headers and footers in preparation for printing.
<u>head</u>	Outputs the first few lines of its input. Useful for getting the header of a file.
<u>tail</u>	Outputs the last few lines of its input. Useful for things like getting the most recent entries from a log file.
<u>tr</u>	Translates characters. Can be used to perform tasks such as upper/lowercase conversions or changing line termination characters from one type to another (for example, converting DOS text files into Unix style text files).
<u>sed</u>	Stream editor. Can perform more sophisticated text translations than <code>tr</code> .
<u>awk</u>	An entire programming language designed for constructing filters. Extremely powerful.

Performing tasks with pipes

1. *Printing from the command line.* Linux provides a program called [lpr](#) that accepts standard input and sends it to the printer. It is often used with pipes

and filters. Here are a couple of examples:

```
cat poorly_formatted_report.txt | fmt | pr | lpr
```

```
cat unsorted_list_with_dupes.txt | sort | uniq | pr | lpr
```

In the first example, we use `cat` to read the file and output it to standard output, which is piped into the standard input of `fmt`. `fmt` formats the text into neat paragraphs and outputs it to standard output, which is piped into the standard input of `pr`. `pr` splits the text neatly into pages and outputs it to standard output, which is piped into the standard input of `lpr`. `lpr` takes its standard input and sends it to the printer.

The second example starts with an unsorted list of data with duplicate entries. First, `cat` sends the list into `sort` which sorts it and feeds it into `uniq` which removes any duplicates. Next `pr` and `lpr` are used to paginate and print the list.

2. *Viewing the contents of tar files* Often you will see software distributed as a *gzipped tar file*. This is a traditional Unix style tape archive file (created with [tar](#)) that has been compressed with [gzip](#). You can recognize these files by their traditional file extensions, ".tar.gz" or ".tgz". You can use the following command to view the directory of such a file on a Linux system:

```
tar tzvf name_of_file.tar.gz | less
```

© 2000-2008, [William Shotts, Jr.](#) Verbatim copying and distribution of this entire article is permitted in any medium, provided this copyright notice is preserved.

Linux® is a registered trademark of Linus Torvalds.