Input and Output

One of the key features of the Linux shell is its modularity. You can forward the output of one command to another. Try the following:

cat /etc/group | sort

This outputs the file group and forwards the output to →*sort*, which sorts the output lines in alphabetical order. You could have run *sort /etc/group* right away, but I wanted to show you how the output of one command can serve as the input of another. Using this output "piping" mechanism (named after the pipe symbol |), you can construct elaborate command lines tailored to the task you want to accomplish.

The group file lists the names of all groups present on your system. Let's say you want to know how many group names start with the letter "s". A newbie would display the file group using *cat* and count the group names starting with "s" by hand. An experienced shell user would execute the following:

grep `^s' /etc/group | wc -l

On my system this outputs the number 10. I don't expect you to understand the details of this command line; the commands *grep* and *wc* will be explained later in this book. In the example, grep outputs all the lines of the file group starting with "s". The output is piped to the wc command, which counts the number of lines in its input when invoked with the -l option. Taken together, the two commands accomplish the given task in an elegant and efficient way.

A shell feature related to piping is the redirection of input and output. Enter the following command:

echo hello

This will output "hello". While *cat* outputs whole files, *echo* is used to output strings. This seems entirely useless, but thanks to the shell's output redirection capabilities you can reroute the output to a file, like this:

echo hello > /tmp/newfile.txt

This writes the string "hello" to the file /tmp/newfile.txt. (Verify this using *cat*!) If the file doesn't exist, as in this case, it is created on the fly. If the file does exist, its contents are deleted before the output is written to it, so be careful with this feature. If you want to append content to an existing file without deleting its previous contents, use >>, like this:

echo "appended text" >> /tmp/newfile.txt

The quotation marks around the string "appended text" help the shell to identify where the string begins and ends; they are optional in this case. You can imagine > and >> as arrows pointing in the direction of the file where the output goes. Input redirection is also possible, using <, but I'm not going to cover such an advanced feature in this short intro.