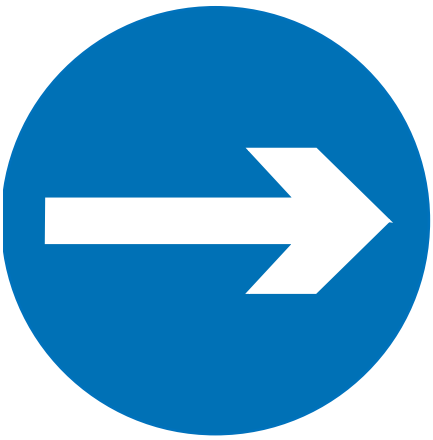

Channels, Pipes and Tee

Redirected

It is quite easy to redirect command line input and output, thus combining commands. This month, we will be looking at the bash operators you need to do this, and showing you how to use tee as an intermediary step.

BY HEIKE JURZIK

For years, "Command Line" has regularly discussed shell commands, letting you in on the techniques needed to skillfully combine programs. We have used redirection operators as a matter of course in these discussions. In this issue of "Command Line" we will be investigating how these operators work under the hood.

Three Channels

There are three standard channels for program input and output. Active progams expect input from standard input, (*stdin*), that is, via the keyboard. Programs send normal output to standard output, (*stdout*), which is typically the terminal window in which you launched the program. There is also a standard error channel, (*stderr*) that typically goes to the same place. This is where to look for error messages issued by a command with notes on how to resolve the problem. So-called **file descriptors** are used to denote the three channels. The standard channels are simply enumerated: *stdin* being #0, *stdout* #1, and *stderr* #2.

The normal program output appears in the terminal window, that is, the prompt, asking us if we really want to run the program. The program reads the response from the standard input channel, that is, from the keyboard. After confirming, an error message appears, telling us that we do not have root privileges. *reiserfsck* sends this message to the standard error output.

The following *grep* command is another straightforward example:

```
hj@asteroid:~$ grep blah *
file1:blah
file2:blah blah
grep: dir: Is a directory
```

In our example, the *grep* command is searching all files for the word "blah". It finds matches in *file1* and *file2*, and in the case of *dir*, it complains that *dir* is a directory rather than a file (more recent *grep* versions simply ignore directories without commenting on the fact, but the Debian Linux *grep* 2.4.2 used for this example, does not accept directories.)

Which Direction?

Programs do not care where commands come from, or where the output goes. Special operators allow you to re-arrange the standard channels. You can use the > operator, to avoid sending standard output to the console:

```
hj@asteroid:~$ grep blah * ➥
> grep_results
grep: dir: Is a directory
```

Instead of the simple > operator, you could type *1*>. As previously mentioned, 1 is the file descriptor assigned to standard output. If you omit the 1, *stdout* is simply assumed as the default. Although the results for this *grep* command are sent to a file called *grep_results*, error messages continue to appear on the console. If you prefer to keep the standard output and redirect the standard error output, try the *2*> operator instead:

```
hj@asteroid:~$ grep blah * ➥
2> grep_error
file1:blah
file2:blah blah
```

If you do not need a program's error output, you can redirect it to */dev/null* instead of into a file. This pseudo-file does not have any content. Any data you send to it is simply discarded. This allows you to remove unwanted error messages:

```
grep blah * 2> /dev/null
```

The > operator not only redirects, it also creates the file needed to do so, or overwrites an existing file with the specified name. As an alternative to this, you can type two greater than signs, >>. If the target file does not exist, it is created. Otherwise, the output is appended to the file.

The >& operator allows you to redirect standard output, and standard error output to a single file:

Command Line

Although GUIs such as KDE or GNOME are useful for various tasks, if you intend to get the most out of your Linux machine, you will need to revert to the good old command line from time to time. Apart from that, you will probably be confronted with various scenarios where some working knowledge will be extremely useful in finding your way through the command line jungle.

GLOSSARY

File descriptor: *Under Linux, each process including open files, has a unique positive identifying number. Each number is also referred to as the file descriptor. The descriptors 0, 1, and 2 are generated automatically, as every process is launched with the standard input, output, and error channels open. See Example 1.*

```
grep blah * >& grep_results
```

Input

Besides redirecting output, you can also redirect input. You need the < operator to do so. For example, a command does not need to read input from the keyboard. It can also parse a file. This is often quite useful for the *mail* command. To send a text, which I have written previously, to the user *petronella*, I simply enter:

```
mail -s "letter" ➥
petronella < lettertext
```

This command first sets the *Subject* of the message, using the *-s* option. *mail* then sends the *lettertext* file which it read using the < operator.

Up the Down Pipe

We can use the pipe character ("|") to link up the input and output streams, thus creating chains of commands. To display the output of the *ls -l /etc/** command in the *less* pager, type :

```
ls -l /etc/* | less
```

Without redirection, the output from the *ls* command would simply scroll out of the terminal window. It is also possible to link up a whole series of pipes, as can be seen in the next example, to pass output through the commands:

```
hj@asteroid:~$ du ~ | ➥
sort -rn | less
15237932 /home/huhn
1598024 /home/huhn/digicam
1518428 /home/huhn/uni
1500236 /home/huhn/tmp
1246740 /home/huhn/uni/english
[...]
```

The *du* command first investigates all the subdirectories in the user's home directory (shown as a tilde ~) to find out how much space the files are occupying. To discover which file is using the most space, you can redirect the output from the *du* command to the *sort* command. The parameters, *-r* and *-n*, ensure that the output will appear in reverse order, and sorted numerically. The second pipe ensures that the output is paged.

Tee Time

The *tee* tool is very useful in commands that use complex pipelines. As the name suggests, *tee* is a kind of t-junction that connects two pipes. *tee* expects data from standard input, which it passes unchanged to standard output, while at the same time copying the data to the specified file. The generic syntax for *tee* is as follows:

```
tee [-option] file
```

You can insert a t-junction into a pipe between two commands. *tee* will accept the output from the first command, redirecting it to a file, and to the next command in the pipeline:

```
hj@asteroid:~$ who | tee ➥
loggedon.txt | grep huhn
huhn     :0     Apr 29 13:45
```

Here, the *who* command first checks who is logged on to the system. The *grep* command checks for instances of the word "huhn". The *tee* command in the middle ensures that the *who* output is first sent to a file called *loggedon.txt*.

tee has two interesting options: *-a* ("append") allows you to append data to an existing file. Without this option, *tee* will simply overwrite the file each time it is called. The *-i* option prevents *tee* from quitting if interrupts ([Ctrl c]) occur in the data stream.

tee is often used if multiple commands occur in a pipeline. Saving the output in temporary files can help troubleshoot and debug complex command chains. Of course, you can remove the *tee* junctions as soon as you have everything working. Logging the output of a command that takes a long time to complete is another useful application for *tee*:

```
tar cvf backup.tar *➥
 | tee backup.log
```

This command creates a logfile in addition to the tar archive, allowing you to check the file at a later stage. Redirecting the *tar* command, as in > *backup.log*, would prevent any output specified by the *v* from cluttering your terminal window. You could still use *tail -f backup.log* to check on *tar*'s progress in a separate terminal window.

Example 1: Cooperating channels

```
01 hj@asteroid:~$ /sbin/fsck /dev/hda7
02 Do you want to run this program?[N/Yes] : Yes
03 reiserfsck: Cannot not open filesystem on "/dev/hda7"
04 Warning... fsck.reiserfs for device /dev/hda7 exited with signal 6.
```