Next Up Previous
: Metacharacters : Shell Programming : Shell Programming

Command Structure

To get a better understanding of the shell and how it works we need to get a better understanding of what a command is and how it is interpreted by the shell.

As noted in chapter two, a command is a series of (white-space delimited) words. The command `echo` prints any arguments to its standard output stream, separated by spaces.

```
$ echo hi there
hi there
```

Commands are usually terminated by a newline, but a semicolon `;` is also a command terminator. Thus typing the line

```
echo the time is ; date
```

to the shell, will cause the two commands to be run one after the other.

```
$ echo the time is ; date
the time is
Wed Mar 1 10:05:00 NZDT 1995
```

If we try sending the output from the line above through a pipe to the `wc` command (forming a *pipeline* command), the result is as follows:

```
$ echo the time is ; date | wc
the time is
       1       6      30
```

It is only the output of the second command which is filtered by `wc`. This is because connecting two commands with a pipe forms a single command. Thus `date | wc` is regarded as a single command which is separated from the command `echo the time is` by `;`.

If it really desired to pipe the output of both commands through wc, the commands can be grouped with parentheses.

```
$ (echo the time is ; date) | wc
       2       9      41
```

Commands can also be terminated by the ampersand character `&`. This works exactly like `;` or a newline, but it tells the shell not to wait for the command to complete before prompting for a new command. Typically this is used to run long running commands ``in the background'' while you continue to type interactive commands. (Now that Unix has a multi-window graphical interface this feature is used much less than it used to be. However, you will

still need to ``background'' jobs if you want them to continue running after you log out.)

```
$ long-running-command &
1525
$
```

The shell prints the process-id number of the command and prompts immediately for another command. The process-id is a unique value which identifies the command for as long as it runs.

The `&` operator terminates commands and because a pipelines are commands, they can be terminated by `&`. This means that we could send the output of a long-running command directly to the printer using the `lpr` command as follows:

```
$ long-running-command | lpr &
```

Most commands accept *arguments* on the command line. These arguments may be the names of files, or a pattern to search for, or an option flag (an argument beginning with `-`). The various special characters interpreted by the shell such as `>`, `<`, `|`, `;` and `&` are *not* arguments to the commands the shell runs. They instead control how the shell runs them. For example, the command

```
$ echo hello > junk
```

tells the shell to redirect the output of the command `echo` into the file `junk`. Neither `>` nor `junk` are arguments to the command `echo`; they are interpreted by the shell and never seen by `echo`. In fact, the command

```
$ > junk echo hello
```

is identical (but less intuitive).

Next Up Previous

: Metacharacters : Shell Programming : Shell Programming

Ross Ihaka *`*