Do not put spaces around the = symbol or the shell will mistake your variable name for a command name.

When referencing a variable, you can surround its name with curly braces to clarify to the parser and to human readers where the variable name stops and other text begins; for example, **${etcdir}** instead of just **$etcdir**. The braces are not normally required, but they can be useful when you want to expand variables inside double-quoted strings. Often, you'll want the contents of a variable to be followed by literal letters or punctuation. For example,

```
$ echo "Saved ${rev}th version of mdadm.conf."
Saved 8th version of mdadm.conf.
```

There's no standard convention for the naming of shell variables, but all-caps names typically suggest environment variables or variables read from global configuration files. More often than not, local variables are all-lowercase with components separated by underscores. Variable names are case sensitive.

Environment variables are automatically imported into **bash**'s variable namespace, so they can be set and read with the standard syntax. Use **export** *varname* to promote a shell variable to an environment variable. Commands for environment variables that you want to set up at login time should be included in your **~/.profile** or **~/.bash_profile** file. Other environment variables, such as PWD for the current working directory, are maintained automatically by the shell.

The shell treats strings enclosed in single and double quotes similarly, except that double-quoted strings are subject to globbing (the expansion of filename-matching metacharacters such as * and ?) and variable expansion. For example:

```
$ mylang="Pennsylvania Dutch"
$ echo "I speak ${mylang}."
I speak Pennsylvania Dutch.
$ echo 'I speak ${mylang}.'
I speak ${mylang}.
```

Back quotes, also known as back-ticks, are treated similarly to double quotes, but they have the additional effect of executing the contents of the string as a shell command and replacing the string with the command's output. For example,

```
$ echo "There are `wc -l < /etc/passwd` lines in the passwd file."
There are 28 lines in the passwd file.
```

Common filter commands

Any well-behaved command that reads STDIN and writes STDOUT can be used as a filter (that is, a component of a pipeline) to process data. In this section we briefly review some of the more widely used filter commands (including some used in passing above), but the list is practically endless. Filter commands are so team oriented that it's sometimes hard to show their use in isolation.

Most filter commands accept one or more filenames on the command line. Only if you fail to specify a file do they read their standard input.