

Table 2.2 shows the **bash** comparison operators for numbers and strings. **bash** uses textual operators for numbers and symbolic operators for strings, exactly the opposite of Perl.

Table 2.2 Elementary bash comparison operators

| String | Numeric | True if |
|-----------------------------------|----------------------|---------------------------------|
| <code>x = y</code> | <code>x -eq y</code> | x is equal to y |
| <code>x != y</code> | <code>x -ne y</code> | x is not equal to y |
| <code>x < y^a</code> | <code>x -lt y</code> | x is less than y |
| – | <code>x -le y</code> | x is less than or equal to y |
| <code>x > y^a</code> | <code>x -gt y</code> | x is greater than y |
| – | <code>x -ge y</code> | x is greater than or equal to y |
| <code>-n x</code> | – | x is not null |
| <code>-z x</code> | – | x is null |

a. Must be backslash-escaped or double bracketed to prevent interpretation as an input or output redirection character.

bash shines in its options for evaluating the properties of files (again, courtesy of its **/bin/test** legacy). Table 2.3 shows a few of **bash**'s many file-testing and file-comparison operators.

Table 2.3 bash file evaluation operators

| Operator | True if |
|------------------------------|--|
| <code>-d file</code> | <i>file</i> exists and is a directory |
| <code>-e file</code> | <i>file</i> exists |
| <code>-f file</code> | <i>file</i> exists and is a regular file |
| <code>-r file</code> | You have read permission on <i>file</i> |
| <code>-s file</code> | <i>file</i> exists and is not empty |
| <code>-w file</code> | You have write permission on <i>file</i> |
| <code>file1 -nt file2</code> | <i>file1</i> is newer than <i>file2</i> |
| <code>file1 -ot file2</code> | <i>file1</i> is older than <i>file2</i> |

Although the `elif` form is useful, a case selection is often a better choice for clarity. Its syntax is shown below in a sample routine that centralizes logging for a script. Of particular note are the closing parenthesis after each condition and the two semicolons that follow the statement block to be executed when a condition is met. The case statement ends with `esac`.

```
# The log level is set in the global variable LOG_LEVEL. The choices
# are, from most to least severe, Error, Warning, Info, and Debug.

function logMsg {
    message_level=$1
    message_itself=$2
```