Comparisons and tests

Flow control in a program is handled by comparison and test statements. Bash also comes with several options to perform tests that are compatible with the Unix system-level features. We can use `if`, `if else`, and logical operators to perform tests and certain comparison operators to compare data items. There is also a command called `test` available to perform tests. Let us see how to use these.

How to do it...

We will have a look at all the different methods used for comparisons and performing tests:

- Using an `if` condition:

  ```
  if condition;
  then
      commands;
  fi
  ```

- Using `else if` and `else`:

  ```
  if condition;
  then
      commands;
  else if condition; then
      commands;
  else
      commands;
  fi
  ```

Nesting is also possible with `if` and `else`. The `if` conditions can be lengthy, to make them shorter we can use logical operators as follows:

- `[ condition ] && action; # action` executes if the condition is true
- `[ condition ] || action; # action` executes if the condition is false

`&&` is the logical AND operation and `||` is the logical OR operation. This is a very helpful trick while writing Bash scripts.

- Performing mathematical comparisons: Usually conditions are enclosed in square brackets `[]`. Note that there is a space between `[` or `]` and operands. It will show an error if no space is provided. An example is as follows:

  ```
  [$var -eq 0 ] or [ $var -eq 0]
  ```

  Performing mathematical conditions over variables or values can be done as follows:

  ```
  [ $var -eq 0 ]  # It returns true when $var equal to 0.
  [ $var -ne 0 ] # It returns true when $var is not equal to 0
  ```

  Other important operators are as follows:

  - `-gt`: Greater than
  - `-lt`: Less than
  - `-ge`: Greater than or equal to
  - `-le`: Less than or equal to

  Multiple test conditions can be combined as follows:

  ```
  [ $var1 -ne 0 -a $var2 -gt 2 ]  # using and -a
  [ $var1 -ne 0 -o var2 -gt 2 ] # OR -o
  ```

- Filesystem related tests: We can test different filesystem-related attributes using different condition flags as follows:

  - `[ -f $file_var ]`: This returns true if the given variable holds a regular file path or filename
  - `[ -x $var ]`: This returns true if the given variable holds a file path or filename that is executable
  - `[ -d $var ]`: This returns true if the given variable holds a directory path or directory name
  - `[ -e $var ]`: This returns true if the given variable holds an existing file
  - `[ -c $var ]`: This returns true if the given variable holds the path of a character device file
  - `[ -b $var ]`: This returns true if the given variable holds the path of a block device file
  - `[ -w $var ]`: This returns true if the given variable holds the path of a file that is writable
  - `[ -r $var ]`: This returns true if the given variable holds the path of a file that is readable
  - `[ -L $var ]`: This returns true if the given variable holds the path of a symlink