Debugging the script

Debugging is one of the critical features that every programming language should implement to produce race-back information when something unexpected happens. Debugging information can be used to read and understand what caused the program to crash or to act in an unexpected fashion. Bash provides certain debugging options that every sysadmin should know. This recipe shows how to use these.

How to do it...

We can either use Bash's inbuilt debugging tools or write our scripts in such a manner that they become easy to debug, here's how:

1. Add the `-x` option to enable debug tracing of a shell script as follows:

   ```
   $ bash -x script.sh
   ```

   Running the script with the `-x` flag will print each source line with the current status. Note that you can also use `sh -x script`.

2. Debug only portions of the script using `set -x` and `set +x`. For example:

   ```
   #!/bin/bash
   #Filename: debug.sh
   for i in {1..6};
   do
       set -x
       echo $i
       set +x
   done
   echo "Script executed"
   ```

   In the preceding script, the debug information for `echo $i` will only be printed, as debugging is restricted to that section using `-x` and `+x`.

3. The aforementioned debugging methods are provided by Bash built-ins. But they always produce debugging information in a fixed format. In many cases, we need debugging information in our own format. We can set up such a debugging style by passing the `_DEBUG` environment variable.

   Look at the following example code:

   ```
   #!/bin/bash
   function DEBUG()
   {
       [ "$_DEBUG" == "on" ] && $@ || :
   }
   ```

```
for i in {1..10}
do
    DEBUG echo $i
done
```

We can run the above script with debugging set to "on" as follows:

```
$ _DEBUG=on ./script.sh
```

We prefix `DEBUG` before every statement where debug information is to be printed. If `_DEBUG=on` is not passed to the script, debug information will not be printed. In Bash, the command `:` tells the shell to do nothing.

How it works...

The `-x` flag outputs every line of script as it is executed to `stdout`. However, we may require only some portions of the source lines to be observed such that commands and arguments are to be printed at certain portions. In such conditions we can use `set builtin` to enable and disable debug printing within the script.

- `set -x`: This displays arguments and commands upon their execution
- `set +x`: This disables debugging
- `set -v`: This displays input when they are read
- `set +v`: This disables printing input

There's more...

We can also use other convenient ways to debug scripts. We can make use of shebang in a trickier way to debug scripts.

Shebang hack

The shebang can be changed from `#!/bin/bash` to `#!/bin/bash -xv` to enable debugging without any additional flags (`-xv` flags themselves).

Functions and arguments

Like any other scripting languages, Bash also supports functions. Let us see how to define and use functions.