2.3. Debugging Bash scripts

2.3.1. Debugging on the entire script

When things don't go according to plan, you need to determine what exactly causes the script to fail. Bash provides extensive debugging features. The most common is to start up the subshell with the `-x` option, which will run the entire script in debug mode. Traces of each command plus its arguments are printed to standard output after the commands have been expanded but before they are executed.

This is the `commented-script1.sh` script ran in debug mode. Note again that the added comments are not visible in the output of the script.

```
willy:~/scripts> bash -x script1.sh
+ clear

+ echo 'The script starts now.'
The script starts now.
+ echo 'Hi, willy!'
Hi, willy!
+ echo

+ echo 'I will now fetch you a list of connected users:'
I will now fetch you a list of connected users:
+ echo

+ w
  4:50pm  up 18 days,  6:49,  4 users,  load average: 0.58, 0.62, 0.40
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty2     -                Sat 2pm  5:36m  0.24s  0.05s  -bash
willy    :0       -                Sat 2pm   ?     0.00s   ?     -
willy    pts/3    -                Sat 2pm 43:13  36.82s 36.82s  BitchX willy ir
willy    pts/2    -                Sat 2pm 43:13   0.13s  0.06s  /usr/bin/screen
+ echo

+ echo 'I'\''m setting two variables now.'
I'm setting two variables now.
+ COLOUR=black
+ VALUE=9
+ echo 'This is a string: '
This is a string:
+ echo 'And this is a number: '
And this is a number:
+ echo

+ echo 'I'\''m giving you back your prompt now.'
I'm giving you back your prompt now.
+ echo
```

There is now a full-fledged debugger for Bash, available at SourceForge. These debugging features are available in most modern versions of Bash, starting from 3.x.

2.3.2. Debugging on part(s) of the script

Using the **set** Bash built-in you can run in normal mode those portions of the script of which you are sure they are without fault, and display debugging information only for troublesome zones. Say we are not sure what the **w** command will do in the example `commented-script1.sh`, then we could enclose it in the script like this:

```
set -x                  # activate debugging from here
w
```

```
set +x                  # stop debugging from here
```

Output then looks like this:

```
willy: ~/scripts> script1.sh
The script starts now.
Hi, willy!

I will now fetch you a list of connected users:

+ w
  5:00pm  up 18 days,  7:00,  4 users,  load average: 0.79, 0.39, 0.33
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty2     -                Sat 2pm  5:47m  0.24s  0.05s  -bash
willy    :0       -                Sat 2pm   ?     0.00s   ?     -
willy    pts/3    -                Sat 2pm 54:02  36.88s 36.88s  BitchX willyke
willy    pts/2    -                Sat 2pm 54:02   0.13s  0.06s  /usr/bin/screen
+ set +x

I'm setting two variables now.
This is a string:
And this is a number:

I'm giving you back your prompt now.

willy: ~/scripts>
```

You can switch debugging mode on and off as many times as you want within the same script.

The table below gives an overview of other useful Bash options:

Table 2-1. Overview of set debugging options

Short notation	Long notation	Result
set -f	set -o noglob	Disable file name generation using metacharacters (globbing).
set -v	set -o verbose	Prints shell input lines as they are read.
set -x	set -o xtrace	Print command traces before executing command.

The dash is used to activate a shell option and a plus to deactivate it. Don't let this confuse you!

In the example below, we demonstrate these options on the command line:

```
willy:~/scripts> set -v

willy:~/scripts> ls
ls
commented-scripts.sh    script1.sh

willy:~/scripts> set +v
set +v

willy:~/scripts> ls *
commented-scripts.sh    script1.sh

willy:~/scripts> set -f

willy:~/scripts> ls *
ls: *: No such file or directory

willy:~/scripts> touch *

willy:~/scripts> ls
*   commented-scripts.sh    script1.sh

willy:~/scripts> rm *

willy:~/scripts> ls
```

```
commented-scripts.sh    script1.sh
```

Alternatively, these modes can be specified in the script itself, by adding the desired options to the first line shell declaration. Options can be combined, as is usually the case with UNIX commands:

#!/bin/bash -xv

Once you found the buggy part of your script, you can add **echo** statements before each command of which you are unsure, so that you will see exactly where and why things don't work. In the example `commented-script1.sh` script, it could be done like this, still assuming that the displaying of users gives us problems:

```
echo "debug message: now attempting to start w command"; w
```

In more advanced scripts, the **echo** can be inserted to display the content of variables at different stages in the script, so that flaws can be detected:

```
echo "Variable VARNAME is now set to $VARNAME."
```
