# Bash flow controls

GNU's bash (Bourne Again Shell, named in punning homage to the Bourne shell and its author, Steven Bourne) is Linux's default shell. Written by Brian Fox and maintained by Chet Ramey, bash's most popular features are its rich command-line editing facilities and its job control abilities. From the programmer's perspective, bash's chief advantages are its customizability and the complete programming environment it provides, including function definitions, integer math, and a surprisingly complete I/O interface. As you might expect of a Linux utility, bash contains elements of all of the popular shells, Bourne, Korn, and C shell, as well as a few innovations of its own. As of this writing, the current release of bash is 4.*. Most Linux distributions, however, use an older version version. In this article we will have a quick look on flow controls.

The few shell scripts presented so far have been fall-through scripts, lacking any control structures such as loops and conditionals. Any programming language worth the name must have facilities for repeating certain code blocks multiple times and for conditionally executing, or not executing, other code blocks. In this section, you will meet all of bash's flow-control structures, which include the following:

- if - Executes one or more statements if a condition is true or false.
- for - Executes one or more statements a fixed number of times.
- while - Executes one or more statements while a condition is true or false.
- until - Executes one or more statements until a condition becomes true or false.
- case - Executes one or more statements depending on the value of a variable.
- select - Executes one or more statements based upon an option selected by a user.

In this article we will have a look only on conditional execution if.

**Conditional execution if**

bash supports conditional execution of code using the if statement, although its evaluation of the condition is slightly different from the behavior of the if statement of languages like C or Pascal. This peculiarity aside, bash's if statement is just as fully featured as C's. Its syntax is summarized below.

*if condition*
*then*
*statements*
*[elif condition statements]*
*[else statements]*
*fi*

First, be sure you understand that if checks the exit status of the last statement in condition. If it is 0 (true), then statements will be executed, but if it is non-zero, the else clause, if present, will be executed and control jumps to the first line of code following fi. The (optional) elif clause(s) (you can have as many as you like) will be executed only if the if condition is false. Similarly, the (optional) else clause will be executed only if all else fails. Generally, Linux programs return 0 on successful or normal completion, and non-zero otherwise, so this limitation is not burdensome.

Regardless of the way programs define their exit codes, bash takes 0 to mean true or normal and non-zero otherwise. If you need specifically to check or save a command's exit code, use the $? operator immediately after running a command. $? returns the exit code of the command most recently run. The matter becomes more complex because bash allows you to combine exit codes in condition using the && and || operators, which approximately translate to logical AND and logical

OR. Suppose that before you enter a code block, you have to change into a directory and copy a file. One way to accomplish this is to use nested ifs, such as in the following code:

```
if cd /home/kwall/data
then
if cp datafile datafile.bak
then
# more code here
fi
fi
```

bash, however, allows you to write this much more concisely, as the following code snippet illustrates:

```
if cd /home/kwall/data && cp datafile datafile.bak
then
# more code here
fi
```

Both code snippets say the same thing, but the second one is much shorter and, in my opinion, much clearer and easier to follow. If, for some reason, the cd command fails, bash will not attempt the copy operation.

Although if only evaluates exit codes, you can use the [...] construct or the bash builtin command, test, to evaluate more complex conditions. [ condition ] returns a code that indicates whether condition is true or false. test does the same thing, but I find it more difficult to read. The range of available conditions that bash automatically provides - currently 35 - is rich and complete. You can test a wide variety of file attributes and compare strings and integers.