Job control in the shell

Nice Job

The right commands make child's play of job control in the shell, allowing you to launch commands in the background, interrupt processes, monitor multiple jobs, and restore specific jobs to the foreground. BY HEIKE JURZIK

In the previous issue of "Command Line", we looked at *screen*, a tool that gives you multiple virtual consoles in a single terminal window. If you do not have *screen*, and the number of consoles you can launch is limited, it makes sense to know how you can launch a program as a background task, interrupt a program, send a program into the background, or restore a program to the foreground. Read on to discover how you can keep control over your jobs.

Into the Background

Let's imagine a situation where a command you have launched is taking a while to complete. You sit there twiddling your thumbs while the terminal waits for the program to complete. And you cannot enter any other commands while you are waiting.

It makes sense to move the process to the background when you are launching it. To do this, just add an ampersand (&) to the command, for example:

```
$ find . -name bla > /tmp/list &
[1] 664
```

This command displays a few messages about the process you launched and then immediately restores the prompt. The message includes the job ID in angled brackets, followed by the PID (**P**rocess **ID**entifier). The PID is unique throughout the system, whereas the job ID is a consecutive number assigned by the shell.

The *jobs* command tells you which jobs are running in the current shell:

```
$ jobs
[1]+  Stopped       find .
-name bla > /tmp/list
```

If you forget to add an ampersand when launching a time-consuming command, and you have the task running in the foreground, you may become very impatient and wish to work on something else. In this situation, the commands to send this task into the background are quite simple in the Bash shell.

First, press [Ctrl-Z] to interrupt the task. The output of the [Ctrl-Z] command tells you the job ID and the name of the command you just interrupted before the shell again displays the prompt. To send the task into the background type *bg* (for "**b**ack**g**round"):

```
$ bg
[1]+ find . -name bla
> /tmp/list &
```

This output displays both the job ID and the complete process name, followed by the ampersand character. And it shows you that the process is now running in the background.

If you have more than one stopped job running in the shell, the *bg* needs details on the process you want to run in the background. By default *bg* moves the process with the highest job ID to the background.

You can again use the *jobs* command to check the job IDs. Then add a percent sign and the job ID to the *bg* command:

```
$ jobs
[1]-  Stopped    find .
-name blub > /tmp/list
[2]+  Stopped    find .
-name bla > /tmp/list
$ bg %2
[2]+ find . -name bla
>/tmp/list &
```

Back to the Foreground

The *fg* ("**f**ore**g**round") command restores a background job to the foreground. Again, the command needs to know which job you mean, for example:

```
fg %2
```

Restoring a job to the foreground means tying up the shell while the foreground process completes. A shell prompt tells you when this is the case:

```
$
[3]-  Done        sleep 5
```

Shortcuts

Bash surprises even experienced users with neat shortcuts and variables that make the administrator's life so much easier.

For example, the exclamation mark can be used as a variable to represent the process ID of the last background process to be launched.

The following commands first display the process ID in addition to the job ID (*jobs -l*), and then run the *echo* command to display the process ID for the last background process to be launched (*$!*):

```
$ sleep 100 &
$ jobs -l
[1]+  1057 Stopped   sleep 1000
[2]   1058 Running   sleep 1000 &
[3]-  1066 Running   sleep 100 &
$ echo $!
1066
```

You can put this information to good use to interrupt the last process. We do not need the *fg* command (the command would need the job ID) and [Ctrl-Z] this time because we will be using the *kill* command instead.

Command Line

Although GUIs such as KDE or GNOME are useful for various tasks, if you intend to get the most out of your Linux machine, you will need to revert to the good old command line from time to time. Apart from that, you will probably be confronted with various scenarios where some working knowledge will be extremely useful in finding your way through the command line jungle.

kill can do more than just shoot processes down in flames; in fact, it can send them all kinds of signals. For an overview, type *kill -l* (or read the manpage *man 7 signal*). In this case, we will be using the *-STOP* signal:

```
$ kill -STOP $!
$ jobs -l
[1]-  1057 Stopped
      sleep 1000
[2]   1058 Running
      sleep 1000 &
[3]+  1066 Stopped
      (signal)
      sleep 100 &
```

Shy Guy

The *nohup* command gives processes the ability to continue running when you quit the shell. To unhitch the process from the shell, type *nohup*, plus the command name, and add an ampersand to send the process to the background. The following output

```
$ nohup sleep 1000 &
[1]   1116 nohup:
appending output to
`nohup.out'
```

tells you that the process will go on running even if you type *exit* or press [Ctrl-D] to quit the shell. Later on, you can check the *nohup.out* file to see what the program did while you were away.

The *nice* command assigns processes a specific priority – this is useful if you have a program running in the background and do not want to risk losing control over the system load. Non-privileged users are only permitted to assign lower priorities to their own tasks.

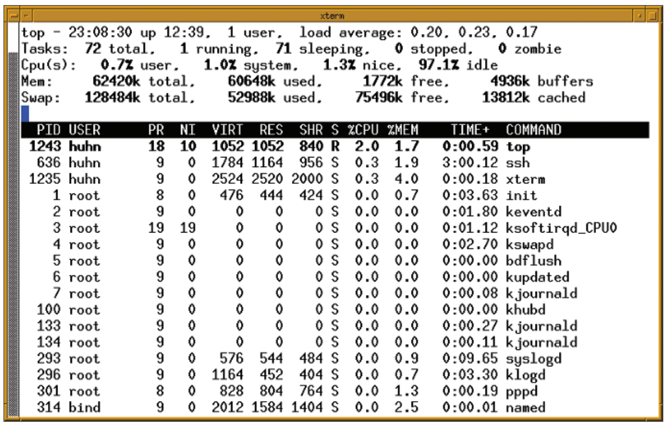
Processes are assigned a *nice* value of 0 by default, where -20 is the highest, and 19 the lowest priority. The *nice* command is followed by the priority and the command, as follows:

```
nice -10 find . ➥
-name bla > /tmp/list
```

If you then call the *ps* command to view the process status, you will notice that the *find* call has been "niced":

```
$ ps auxwww
[...]
huhn   1200  0.2  0.6
1520 404 pts/7 RN+
23:02 0:00 find .  -
name bla [...]
```

The *top* command gives you an even better way of discovering a program's *nice* level. The *top* program lists processes, and sorts them by CPU usage. The fourth column, the one with the *NI* heading, tells you the *nice* level (Figure 1).



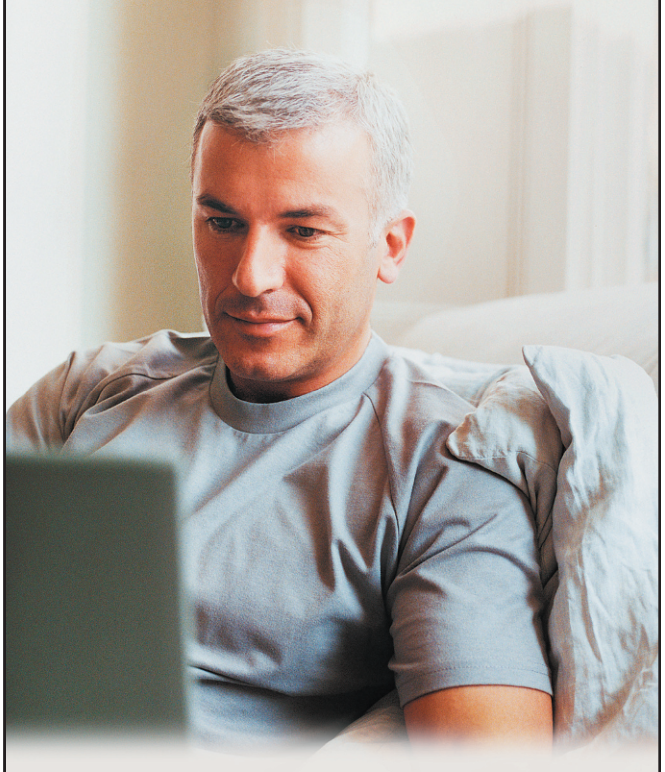
```
top - 23:08:30 up 12:39,  1 user,  load average: 0.20, 0.23, 0.17
Tasks:  72 total,   1 running,  71 sleeping,   0 stopped,   0 zombie
Cpu(s):   0.7% user,   1.0% system,   1.3% nice,  97.1% idle
Mem:      62420k total,    60648k used,     1772k free,     4936k buffers
Swap:    128484k total,    52988k used,    75496k free,    13812k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1243 huhn      18  10  1052 1052  840 R  2.0  1.7   0:00.59 top
  636 huhn       9   0  1784 1164  956 S  0.3  1.9   3:00.12 ssh
 1235 huhn       9   0  2524 2520 2000 S  0.3  4.0   0:00.18 xterm
    1 root       8   0   476  444  424 S  0.0  0.7   0:03.63 init
    2 root       9   0     0    0    0 S  0.0  0.0   0:01.80 keventd
    3 root      19  19     0    0    0 S  0.0  0.0   0:01.12 ksoftirqd_CPU0
    4 root       9   0     0    0    0 S  0.0  0.0   0:02.70 kswapd
    5 root       9   0     0    0    0 S  0.0  0.0   0:00.00 bdflush
    6 root       9   0     0    0    0 S  0.0  0.0   0:00.00 kupdated
    7 root       9   0     0    0    0 S  0.0  0.0   0:00.08 kjournald
  100 root       9   0     0    0    0 S  0.0  0.0   0:00.00 khubd
  133 root       9   0     0    0    0 S  0.0  0.0   0:00.27 kjournald
  134 root       9   0     0    0    0 S  0.0  0.0   0:00.11 kjournald
  293 root       9   0   576  544  484 S  0.0  0.9   0:09.65 syslogd
  296 root       9   0  1164  452  404 S  0.0  0.7   0:03.30 klogd
  301 root       8   0   828  804  764 S  0.0  1.3   0:00.19 pppd
  314 bind       9   0  2012 1584 1404 S  0.0  2.5   0:00.01 named
```

Figure 1: The "top" command also tells you a program's "nice" level.

OPERA software

elax and enjoy the Web



Opera browser - a safer and faster Internet experience.

- op-up iller
- Integrated search
- Tabbed browsing
- Spam-filtered e-mail
- Advanced security
- Cross-platform

Get Opera for Linux at: www.opera.com/linuxmag