Linux Administration

Managing processes

Xavier Belanger

Definitions

- A process is a program running in memory, and using other resources as needed (network, graphical display, etc.).
- The main states for a process are: running, waiting or blocked, and then terminated.
- Processes are linked to a user, and all related to the main system process: init.

Checking processes

- *ps* is the most versatile command to check on processes.
- *pstree* can give you a “visual” representation of all processes (parent/child).
- *pidof* will list the PID of a given executable name.
- *top* is an interactive and real-time view of all processes running.

The ps command

- Without any option, *ps* will display all processes for your current shell.
- *ps ux* will display all your processes, across multiple shells.
- *ps aux* will display all processes for all users.
- *ps u -u <username>* will display all processes for a specific user account.
- *ps u -C <process name>* will display information only for a specific application.

The pstree command

- By default the *pstree* output includes all processes running on the system.
- You can add the *-p* option to display the PID for each process.
- *pstree <username>* will restrict the output to a specific user.

The pidof command

- pidof gives the PID of a given application, or nothing if no match is found.
- If you are searching of a specific application, you should provide the file full name.

The top command

- *top* will display a list of all processes running on the system, plus some basic information with an automatic refresh (every three seconds by default).
- Use the ‘*q*’ key to quit the command.

Managing processes

- If a program becomes unresponsive, using too much resources or otherwise causing issues, you can terminate it with the *kill* command.
- *kill* requires the process identifier (PID) to target the proper process; you can obtain the PID with the *ps* command.
- *killall* is another command that can be used to send a signal to multiple processes with the same name.
- *nice* can be use to change a process priority.

The kill command

- *kill -l* will list all type of signals that you can send to a process.
- By default the TERM signal is sent to terminate the targeted process.
- Other signals can be used; but the result will depend on how the targeted program has been set to process a given signal (some can be ignored, others cannot).

The killall command

- The *killall* command works is a similar way as the *kill* one, the main difference is that multiple processes could be impacted.
- *killall -i* will ask for a confirmation before terminating each process.

The nice command

- A regular process starts with a default priority value of 0. That value can be modified between -20 (highest priority) and 19 (lowest priority).
- *nice* can change that value. If the priority need to be modified again for the same process, use the *renice* command.
- Only the root user can assign a priority below zero.

Communication channels

Processes have three channels available by default for communication:

- Standard Input (stdin, code 0)
- Standard Output (stdout, code 1)
- Standard Error (stderr, code 2)

Channel redirection

- You can redirect specific channels from or to a process by using the greater than (>) and lesser than (<) symbols.
- Redirecting the standard output to a file:

  find /etc -type f -name ‘a’> ~/etc-a-files*

Specific redirections

- Adding (not overwriting) to an existing file:

  ```
  find /etc -type f -name 'a*' >> ~/etc-a-files
  ```

- Redirecting only the error channel:

  ```
  find /etc -type f -name 'a*' 2> ~/etc-a-files-err
  ```

- Redirecting both standard and error channels to the same file:

  ```
  find /etc -type f -name 'a*' > ~/etc-a-files-both 2>&1
  ```

Pipes

- The vertical bar symbol (|) can be used to combine the standard output from one command to be used as the standard input for the following one.
- *command_1* | *command_2*
- You can combine more than two commands that way.

Combining commands

- You can use the AND and OR logical operators to combine commands.
- With AND, the second command is executed only if the first one is successful.

  command_1 && command_2
- With OR, the second command is executed only if the first one fails.

  command_1 || command_2

Command substitution

- You can obtain the result of a command to be processed by another one using the $(…) syntax:
- *cat $(find /etc -type f -name 'a*')*
- The command inside the parentheses will be executed first and the result will be used by the initial command.