0x05. Processes and signals

DevOps | Shell | Bash | Syscall | Scripting

Weight: 1

Project over - took place from Nov 24, 2023 6:00 AM to Nov 25, 2023 6:00 AM

An auto review will be launched at the deadline

In a nutshell…

- **Auto QA review:** 26.0/26 mandatory & 0.0/17 optional
- **Altogether: 100.0%**
  - Mandatory: 100.0%
  - Optional: 0.0%
  - Calculation: 100.0% + (100.0% * 0.0%) == **100.0%**

About Bash projects

Unless stated, all your projects will be auto-corrected with Ubuntu 20.04 LTS.

Resources

Read or watch:

- Linux PID (/rltoken/qVGxUt1QMIV4B4oVrQBlQg)
- Linux process (/rltoken/px2TdWSjVO8i9SB5gHchAw)
- Linux signal (/rltoken/qQSGz9CN52PVF3IPCuaRiw)

- Process management in linux (/rltoken/XlYrlghzNZ6Z1cbl_lPaiA)

(/)

man or help:

- `ps`
- `pgrep`
- `pkill`
- `kill`
- `exit`
- `trap`

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/_zeQBWHdlNNOM-5IqFDhSQ), **without the help of Google**:

General

- What is a PID
- What is a process
- How to find a process' PID
- How to kill a process
- What is a signal
- What are the 2 signals that cannot be ignored

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be interpreted on Ubuntu 20.04 LTS
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project, is mandatory
- All your Bash script files must be executable
- Your Bash script must pass `Shellcheck` (version `0.7.0` via `apt-get`) without any error
- The first line of all your Bash scripts should be exactly `#!/usr/bin/env bash`
- The second line of all your Bash scripts should be a comment explaining what is the script doing

More Info

For those who want to know more and learn about all signals, check out this article (/ritoken/BOU-KVNMqfKElBo_VOI26A).

Tasks

0. What is my PID

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a Bash script that displays its own PID.

```
sylvain@ubuntu$ ./0-what-is-my-pid
4120
sylvain@ubuntu$
```

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `0-what-is-my-pid`

Done! | Check your code | Get a sandbox | QA Review

1. List your processes

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a Bash script that displays a list of currently running processes.

Requirements:

- Must show all processes, for all users, including those which might not have a TTY
- Display in a user-oriented format
- Show process hierarchy

(7)

```
sylvain@ubuntu$ ./1-list_your_processes | head -50
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         2  0.0  0.0      0     0 ?        S    Feb13   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [ksoftirqd/0]
root         4  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [kworker/0:0]
root         5  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    Feb13   0:02  \_ [rcu_sched]
root         8  0.0  0.0      0     0 ?        S    Feb13   0:03  \_ [rcuos/0]
root         9  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [rcu_bh]
root        10  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [rcuob/0]
root        11  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [migration/0]
root        12  0.0  0.0      0     0 ?        S    Feb13   0:02  \_ [watchdog/0]
root        13  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [khelper]
root        14  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [kdevtmpfs]
root        15  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [netns]
root        16  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [writeback]
root        17  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [kintegrityd]
root        18  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [bioset]
root        19  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [kworker/u3:0]
root        20  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [kblockd]
root        21  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [ata_sff]
root        22  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [khubd]
root        23  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [md]
root        24  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [devfreq_wq]
root        25  0.0  0.0      0     0 ?        S    Feb13   0:41  \_ [kworker/0:1]
root        27  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [khungtaskd]
root        28  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [kswapd0]
root        29  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [vmstat]
root        30  0.0  0.0      0     0 ?        SN   Feb13   0:00  \_ [ksmd]
root        31  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [fsnotify_mark]
root        32  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [ecryptfs-kthrea]
root        33  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [crypto]
root        45  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [kthrotld]
root        46  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [kworker/u2:1]
root        65  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [deferwq]
root        66  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [charger_manager]
root       108  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [kpsmoused]
root       125  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [scsi_eh_0]
root       126  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [kworker/u2:2]
root       172  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [jbd2/sda1-8]
root       173  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [ext4-rsv-conver]
root       409  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [iprt]
root       549  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [kworker/u3:1]
root       808  0.0  0.0      0     0 ?        S    Feb13   0:00  \_ [kauditd]
root       834  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [rpciod]
root       846  0.0  0.0      0     0 ?        S<   Feb13   0:00  \_ [nfsiod]
root         1  0.0  0.4  33608  2168 ?        Ss   Feb13   0:00 /sbin/init
root       373  0.0  0.0  19472   408 ?        S    Feb13   0:00 upstart-udev-bridge --daemo
n
root       378  0.0  0.2  49904  1088 ?        Ss   Feb13   0:00 /lib/systemd/systemd-udevd
--daemon
root       518  0.0  0.1  23416   644 ?        Ss   Feb13   0:00 rpcbind
```

```
statd      547  0.0  0.1  21536   852 ?        Ss   Feb13   0:00 rpc.statd -L
sylvain@ubuntu$
```

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `1-list_your_processes`

Done! | Check your code | Get a sandbox | QA Review

2. Show your Bash PID

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Using your previous exercise command, write a Bash script that displays lines containing the `bash` word, thus allowing you to easily get the PID of your Bash process.

Requirements:

- You cannot use `pgrep`
- The third line of your script must be `# shellcheck disable=SC2009` (for more info about ignoring `shellcheck` error here (/rltoken/vErRT8QGU2bwJ6FLvPLzxw))

```
sylvain@ubuntu$ sylvain@ubuntu$ ./2-show_your_bash_pid
sylvain   4404  0.0  0.7  21432  4000 pts/0    Ss   03:32   0:00          \_ -bash
sylvain   4477  0.0  0.2  11120  1352 pts/0    S+   03:40   0:00              \_ bash ./2-sh
ow_your_bash_PID
sylvain   4479  0.0  0.1  10460   912 pts/0    S+   03:40   0:00                  \_ grep ba
sh
sylvain@ubuntu$
```

Here we can see that my Bash PID is `4404`.

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `2-show_your_bash_pid`

Done! | Check your code | Get a sandbox | QA Review

3. Show your Bash PID made easy

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a Bash script that displays the PID, along with the process name, of processes whose name contain the word `bash`.

Requirements:

- You cannot use `ps`

```
sylvain@ubuntu$ ./3-show_your_bash_pid_made_easy
4404 bash
4555 bash
sylvain@ubuntu$ ./3-show_your_bash_pid_made_easy
4404 bash
4557 bash
sylvain@ubuntu$
```

Here we can see that:

- For the first iteration: `bash` PID is `4404` and that the `3-show_your_bash_pid_made_easy` script PID is `4555`
- For the second iteration: `bash` PID is `4404` and that the `3-show_your_bash_pid_made_easy` script PID is `4557`

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `3-show_your_bash_pid_made_easy`

Done! | Check your code | Get a sandbox | QA Review

4. To infinity and beyond

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a Bash script that displays `To infinity and beyond` indefinitely.

Requirements:

- In between each iteration of the loop, add a `sleep 2`

```
sylvain@ubuntu$ ./4-to_infinity_and_beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
^C
sylvain@ubuntu$
```

Note that I `ctrl+c` (killed) the Bash script in the example.

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `4-to_infinity_and_beyond`

Done! | Check your code | Get a sandbox | QA Review

5. Don't stop me now!

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

We stopped our `4-to_infinity_and_beyond` process using `ctrl+c` in the previous task, there is actually another way to do this.

Write a Bash script that stops `4-to_infinity_and_beyond` process.

Requirements:

- You must use `kill`

Terminal #0

```
sylvain@ubuntu$ ./4-to_infinity_and_beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
Terminated
sylvain@ubuntu$
```

Terminal #1

```
sylvain@ubuntu$ ./5-dont_stop_me_now
sylvain@ubuntu$
```

I opened 2 terminals in this example, started by running my `4-to_infinity_and_beyond` Bash script in terminal #0 and then moved on terminal #1 to run `5-dont_stop_me_now`. We can then see in terminal #0 that my process has been terminated.

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `5-dont_stop_me_now`

Done! | Check your code | Get a sandbox | QA Review

6. Stop me if you can

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a Bash script that stops `4-to_infinity_and_beyond` process.

Requirements:

- You cannot use `kill` or `killall`

Terminal #0

(7)

```
sylvain@ubuntu$ ./4-to_infinity_and_beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
Terminated
sylvain@ubuntu$
```

Terminal #1

```
sylvain@ubuntu$ ./6-stop_me_if_you_can
sylvain@ubuntu$
```

I opened 2 terminals in this example, started by running my `4-to_infinity_and_beyond` Bash script in terminal #0 and then moved on terminal #1 to run `6-stop_me_if_you_can`. We can then see in terminal #0 that my process has been terminated.

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `6-stop_me_if_you_can`

Done! | Check your code | Get a sandbox | QA Review

7. Highlander

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a Bash script that displays:

- `To infinity and beyond` indefinitely
- With a `sleep 2` in between each iteration
- `I am invincible!!!` when receiving a `SIGTERM` signal

Make a copy of your `6-stop_me_if_you_can` script, name it `67-stop_me_if_you_can`, that kills the `7-highlander` process instead of the `4-to_infinity_and_beyond` one.

Terminal #0

(7)

```
sylvain@ubuntu$ ./7-highlander
To infinity and beyond
To infinity and beyond
I am invincible!!!
To infinity and beyond
I am invincible!!!
To infinity and beyond
To infinity and beyond
To infinity and beyond
I am invincible!!!
To infinity and beyond
^C
sylvain@ubuntu$
```

Terminal #1

```
sylvain@ubuntu$ ./67-stop_me_if_you_can
sylvain@ubuntu$ ./67-stop_me_if_you_can
sylvain@ubuntu$ ./67-stop_me_if_you_can
sylvain@ubuntu$
```

I started `7-highlander` in Terminal #0 and then run `67-stop_me_if_you_can` in terminal #1, for every iteration we can see `I am invincible!!!` appearing in terminal #0.

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `7-highlander`

Done! | Check your code | Get a sandbox | QA Review

8. Beheaded process

mandatory

Score: 100.0% (*Checks completed: 100.0%*)

Write a Bash script that kills the process `7-highlander`.

Terminal #0

```
sylvain@ubuntu$ ./7-highlander
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
Killed
sylvain@ubuntu$
```

Terminal #1

```
sylvain@ubuntu$ ./8-beheaded_process
sylvain@ubuntu$
```

I started `7-highlander` in Terminal #0 and then run `8-beheaded_process` in terminal #1 and we can see that the `7-highlander` has been killed.

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `8-beheaded_process`

Done! | Check your code | Get a sandbox | QA Review

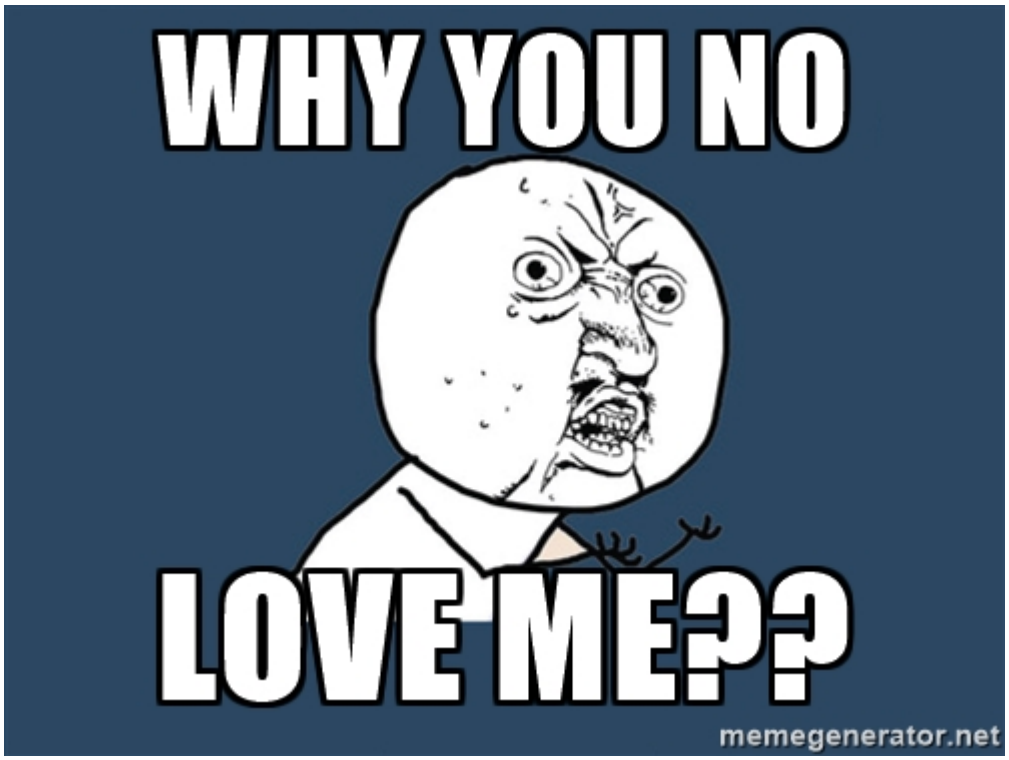
9. Process and PID file

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Write a Bash script that:

- Creates the file `/var/run/myscript.pid` containing its PID
- Displays `To infinity and beyond` indefinitely
- Displays `I hate the kill command` when receiving a SIGTERM signal
- Displays `Y U no love me?!` when receiving a SIGINT signal
- Deletes the file `/var/run/myscript.pid` and terminates itself when receiving a SIGQUIT or SIGTERM signal



(7)

```
sylvain@ubuntu$ sudo ./100-process_and_pid_file
To infinity and beyond
To infinity and beyond
^CY U no love me?!
```

Executing the `100-process_and_pid_file` script and killing it with `ctrl+c`.

Terminal #0

```
sylvain@ubuntu$ sudo ./100-process_and_pid_file
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
To infinity and beyond
I hate the kill command
sylvain@ubuntu$
```

Terminal #1

```
sylvain@ubuntu$ sudo pkill -f 100-process_and_pid_file
sylvain@ubuntu$
```

Starting `100-process_and_pid_file` in the terminal #0 and then killing it in the terminal #1.

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `100-process_and_pid_file`

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review

10. Manage my process

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

Read:

- & (/rltoken/R4YSgPT1k0PhWCrB0TYzoQ)
- init.d (/rltoken/sVqN4oNYYO6ojS4ctT02Jw)
- Daemon (/rltoken/kCoQ5aYO3towdDQFVPcfNg)
- Positional parameters (/rltoken/TJ2rxUwRsnM1mJQHSCnOQA)

man: `sudo`

Programs that are detached from the terminal and running in the background are called daemons or processes, need to be managed. The general minimum set of instructions is: `start` , `restart` and `stop` . The most popular way of doing so on Unix system is to use the init scripts.

Write a `manage_my_process` Bash script that:

- Indefinitely writes `I am alive!` to the file `/tmp/my_process`
- In between every `I am alive!` message, the program should pause for 2 seconds

Write Bash (init) script `101-manage_my_process` that manages `manage_my_process` . (both files need to be pushed to git)

Requirements:

- When passing the argument `start` :
  - Starts `manage_my_process`
  - Creates a file containing its PID in `/var/run/my_process.pid`
  - Displays `manage_my_process started`
- When passing the argument `stop` :
  - Stops `manage_my_process`
  - Deletes the file `/var/run/my_process.pid`
  - Displays `manage_my_process stopped`
- When passing the argument `restart`
  - Stops `manage_my_process`
  - Deletes the file `/var/run/my_process.pid`
  - Starts `manage_my_process`

(/)

- Creates a file containing its PID in `/var/run/my_process.pid`
- Displays `manage_my_process restarted`

- Displays `Usage: manage_my_process {start|stop|restart}` if any other argument or no argument is passed

Note that this init script is far from being perfect (but good enough for the sake of manipulating process and PID file), for example we do not handle the case where we check if a process is already running when doing `./101-manage_my_process start`, in our case it will simply create a new process instead of saying that it is already started.

```
sylvain@ubuntu$ sudo ./101-manage_my_process
Usage: manage_my_process {start|stop|restart}
sylvain@ubuntu$ sudo ./101-manage_my_process start
manage_my_process started
sylvain@ubuntu$ tail -f -n0 /tmp/my_process
I am alive!
I am alive!
I am alive!
I am alive!
^C
sylvain@ubuntu$ sudo ./101-manage_my_process stop
manage_my_process stopped
sylvain@ubuntu$ cat /var/run/my_process.pid
cat: /var/run/my_process.pid: No such file or directory
sylvain@ubuntu$ tail -f -n0 /tmp/my_process
^C
sylvain@ubuntu$ sudo ./101-manage_my_process start
manage_my_process started
sylvain@ubuntu$ cat /var/run/my_process.pid
11864
sylvain@ubuntu$ sudo ./101-manage_my_process restart
manage_my_process restarted
sylvain@ubuntu$ cat /var/run/my_process.pid
11918
sylvain@ubuntu$ tail -f -n0 /tmp/my_process
I am alive!
I am alive!
I am alive!
^C
sylvain@ubuntu$
```

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `101-manage_my_process, manage_my_process`

11. Zombie

#advanced

Score: 0.0% (*Checks completed: 0.0%*)

(http://fineartamerica.com/featured/zombie-seahorse-lauren-b.html)

Read what a zombie process is (/rltoken/Tb86ZoSxR6ORCKYlZaYzHw).

Write a C program that creates 5 zombie processes.

Requirements:

- For every zombie process created, it displays `Zombie process created, PID: ZOMBIE_PID`
- Your code should use the Betty style. It will be checked using `betty-style.pl` and `betty-doc.pl`

- When your code is done creating the parent process and the zombies, use the function bellow (/)

```
int infinite_while(void)
{
    while (1)
    {
        sleep(1);
    }
    return (0);
}
```

Example:

Terminal #0

```
sylvain@ubuntu$ gcc 102-zombie.c -o zombie
sylvain@ubuntu$ ./zombie
Zombie process created, PID: 13527
Zombie process created, PID: 13528
Zombie process created, PID: 13529
Zombie process created, PID: 13530
Zombie process created, PID: 13531
^C
sylvain@ubuntu$
```

Terminal #1

```
sylvain@ubuntu$ ps aux | grep -e 'Z+.*<defunct>'
sylvain  13527  0.0  0.0      0     0 pts/0    Z+   01:19   0:00 [zombie] <defunct>
sylvain  13528  0.0  0.0      0     0 pts/0    Z+   01:19   0:00 [zombie] <defunct>
sylvain  13529  0.0  0.0      0     0 pts/0    Z+   01:19   0:00 [zombie] <defunct>
sylvain  13530  0.0  0.0      0     0 pts/0    Z+   01:19   0:00 [zombie] <defunct>
sylvain  13531  0.0  0.0      0     0 pts/0    Z+   01:19   0:00 [zombie] <defunct>
sylvain  13533  0.0  0.1  10460   964 pts/2    S+   01:19   0:00 grep --color=auto -e Z+.*<d
efunct>
sylvain@ubuntu$
```

In Terminal #0, I start by compiling `102-zombie.c` and executing `zombie` which creates 5 zombie processes. In Terminal #1, I display the list of processes and look for lines containing `Z+.*<defunct>` which catches zombie process.

Repo:

- GitHub repository: `alx-system_engineering-devops`
- Directory: `0x05-processes_and_signals`
- File: `102-zombie.c`

Done? | Check your code | Ask for a new correction | Get a sandbox | QA Review