

1. LINUX COMMANDS TO KILL PROCESSES

(i) Kill processes by name

👉 If Adam wants to kill all processes having the same name, he can use:

Command:

```
bash  
  
killall process_name
```

Example:

```
bash  
  
killall firefox
```

✳️ This command stops all running Firefox processes at once.

(ii) Kill a process based on the process name

👉 If Adam knows the process name but wants more control, he can use:

Command:

```
bash  
  
pkill process_name
```

Example:

```
bash  
  
pkill chrome
```

✳️ This kills processes by matching the process name.

It is very useful when multiple processes are running.

(iii) Kill a single process at a time using Process ID (PID)

👉 First, Adam must find the **PID** of the process, then kill it.

Step 1: Find PID

```
bash
```

```
ps
```

or

```
bash
```

```
ps -ef
```

Step 2: Kill the process using PID

```
bash
```

```
kill PID
```

Example:

```
bash
```

```
kill 2456
```

✳️ This command kills **only one specific process**.

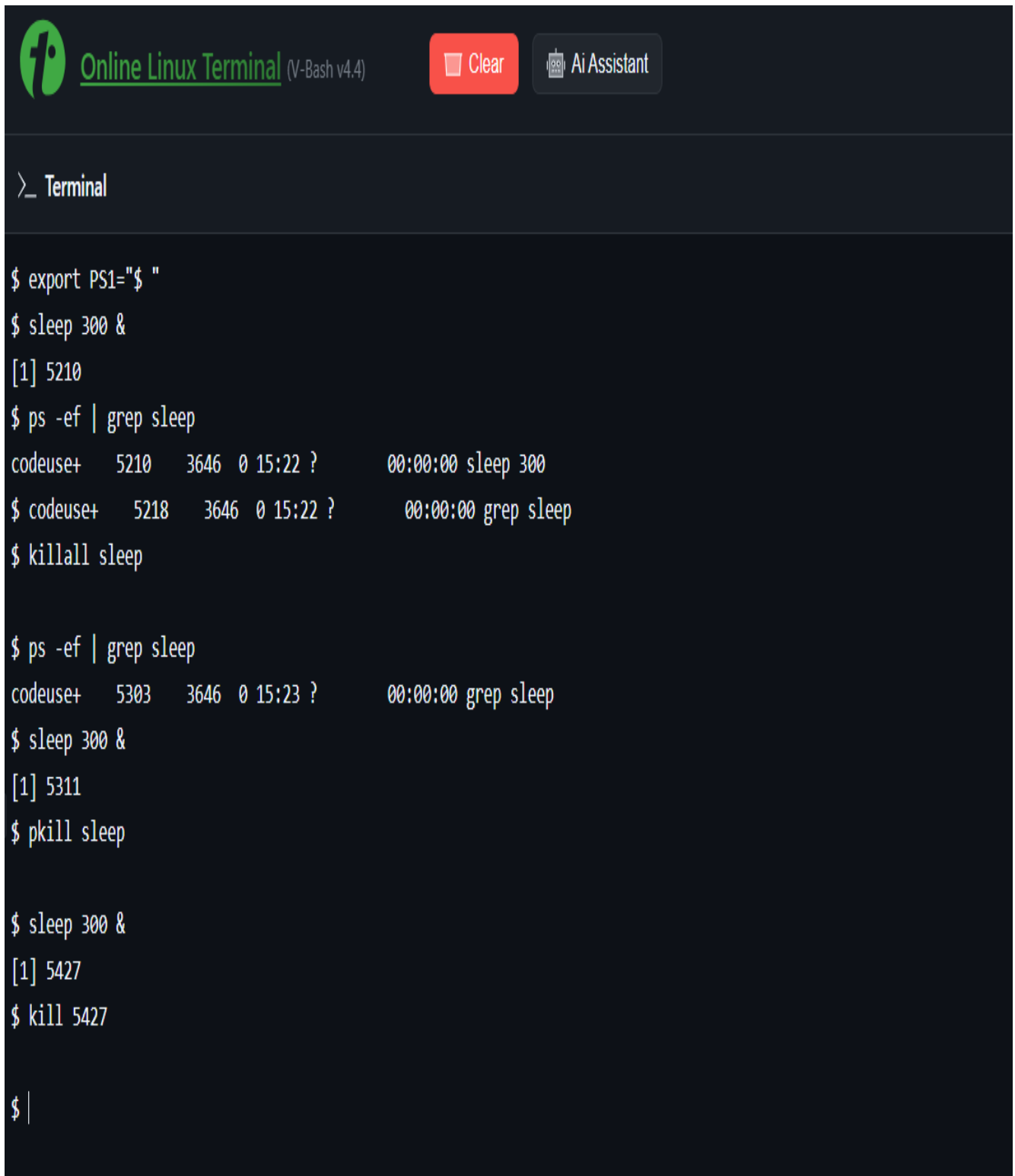
OPERATION

- Kill processes by name
- Kill process based on name
- Kill single process using PID

COMMAND USED

- killall process_name
- pkill process_name
- kill PID

OUTPUT :



The image shows a screenshot of an "Online Linux Terminal" window. At the top, there is a header bar with a green logo on the left, the text "Online Linux Terminal (V-Bash v4.4)" in the center, and two buttons on the right: a red "Clear" button and a grey "Ai Assistant" button. Below the header, the terminal content is displayed on a dark background with light green text. The terminal shows a series of commands and their outputs. The first command is "export PS1=\"\$ " which sets the prompt. Then, "sleep 300 &" is run, and the prompt changes to "[1] 5210". The user then runs "ps -ef | grep sleep", which shows a process table with columns for user, PID, PPID, priority, time, and command. The output shows a "sleep 300" process owned by "codeuse+" with PID 5210. Next, the user runs "killall sleep", and the prompt returns to "\$". Then, the user runs "ps -ef | grep sleep" again, showing the "grep sleep" process. Finally, the user runs "sleep 300 &" again, and the prompt changes to "[1] 5311". The user then runs "pkill sleep", and the prompt returns to "\$". The terminal ends with the user running "sleep 300 &" again, and the prompt changes to "[1] 5427". The user then runs "kill 5427", and the prompt returns to "\$". The terminal is currently at the "\$" prompt with a cursor.

```
>_ Terminal

$ export PS1="$ "
$ sleep 300 &
[1] 5210
$ ps -ef | grep sleep
codeuse+  5210   3646  0 15:22 ?        00:00:00 sleep 300
$ codeuse+  5218   3646  0 15:22 ?        00:00:00 grep sleep
$ killall sleep

$ ps -ef | grep sleep
codeuse+  5303   3646  0 15:23 ?        00:00:00 grep sleep
$ sleep 300 &
[1] 5311
$ pkill sleep

$ sleep 300 &
[1] 5427
$ kill 5427

$ |
```

2.A) ORPHAN PROCESS:

An orphan process is a child process whose parent process terminates before the child finishes execution. The orphan process is adopted by the init process.

```
c
#include <stdio.h>
#include <unistd.h>

int main() {
    int pid = fork();

    if (pid > 0) {
        printf("Parent process exiting\n");
    } else {
        sleep(5);
        printf("Orphan Child Process\n");
        printf("PID: %d\n", getpid());
        printf("PPID: %d\n", getppid());
    }
    return 0;
}
```

OUTPUT:

```
Parent process exiting
Orphan Child Process
PID: 4321
PPID: 1
```

(B) ZOMBIE PROCESS:

A zombie process is a process that has completed execution but still has an entry in the process table because its parent has not read its exit status.

```
c
#include <stdio.h>
#include <unistd.h>

int main() {
    int pid = fork();

    if (pid == 0) {
        printf("Child process exiting\n");
    } else {
        sleep(10);
        printf("Parent process running\n");
    }
    return 0;
}
```

OUTPUT:

```
Child process exiting
Parent process running
```

3.(A) CHILD PROCESS:

A child process is a process that is created by another process using the fork () system call. The child process gets a new PID and executes independently of the parent process.

(B) PARENT PROCESS:

A parent process is the process that creates another process using the fork () system call. It receives the PID of the child process as the return value of fork ().

➤ **IMP:**

- ❖ fork () returns **0** → Child process
- ❖ fork () returns **PID > 0** → Parent process

```
#include <stdio.h>
#include <unistd.h>

int main() {
    int pid = fork();

    if (pid == 0) {
        printf("Child Process\n");
        printf("PID: %d\n", getpid());
        printf("PPID: %d\n", getppid());
    } else {
        printf("Parent Process\n");
        printf("PID: %d\n", getpid());
        printf("Child PID: %d\n", pid);
    }
    return 0;
}
```

OUTPUT:

```
Child Process
PID: 4501
PPID: 4500
Parent Process
PID: 4500
Child PID: 4501
```