Linux commands to kill processes

Adam can reduce system load using the following commands:

1. Kill processes by name

pkill firefox

👉 Kills **all processes** with the name *firefox*.

---

2. Kill a process based on process name

killall chrome

👉 Terminates all running instances of *chrome*.

---

3. Kill a single process using Process ID (PID)

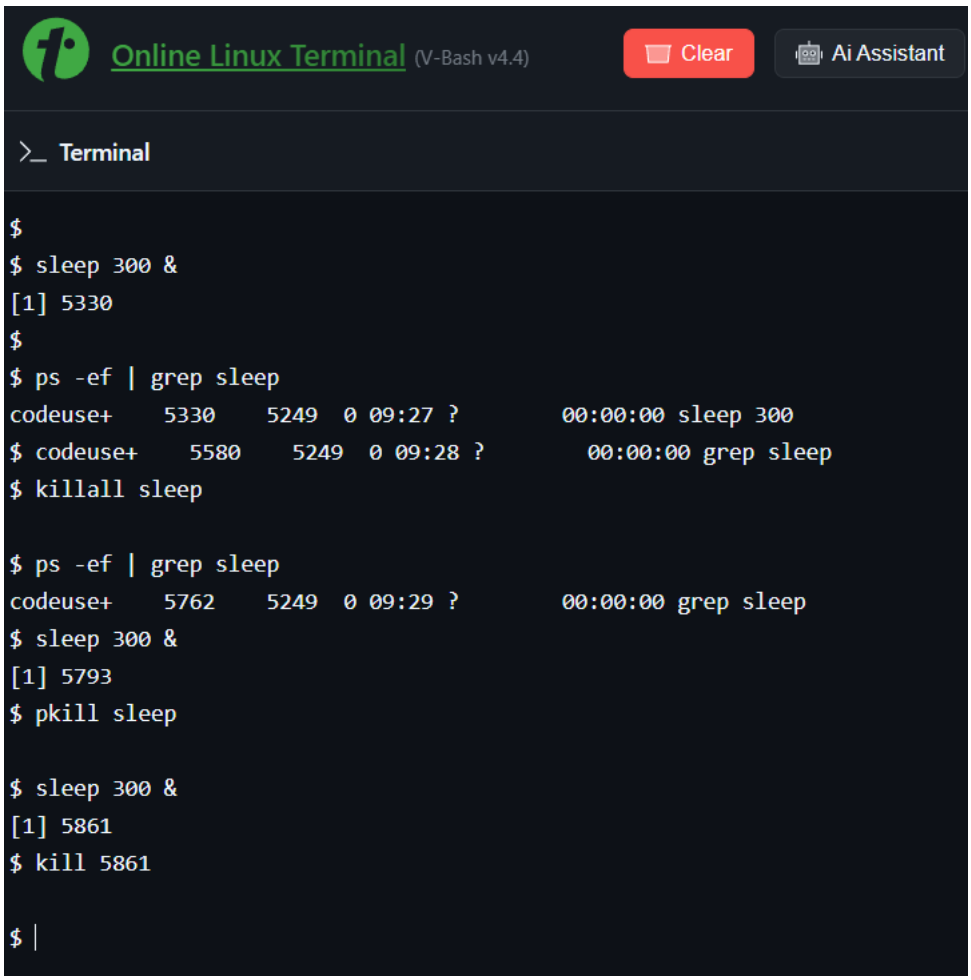
kill 1234

👉 Kills the process with **PID = 1234**.

Force kill (if not responding):

kill -9 1234

OUTPUT:



Orphan Process

An orphan process is a child process whose parent process has terminated.
It is automatically adopted by the init/systemd process (PID 1).

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

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

    if (pid > 0) {
        // Parent process
        printf("Parent process exiting...\n");
    }
    else if (pid == 0) {
        // Child process
        sleep(5);
        printf("Child process running...\n");
        printf("Child PID: %d\n", getpid());
        printf("New Parent PID: %d\n", getppid());
    }

    return 0;
}
```

Output:

```
Parent process exiting...
Child process running...
Child PID: 3456
New Parent PID: 1
```

Zombie Process

A zombie process is a child process that has completed execution but still has an entry in the process table.
This happens when the parent does not call wait() to collect its exit status.

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

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

    if (pid > 0) {
        // Parent process
        sleep(10);   // Parent sleeps, does not wait
        printf("Parent process running...\n");
    }
    else if (pid == 0) {
        // Child process
        printf("Child process exiting...\n");
    }

    return 0;
}
```

Output:

```
0 Z root   4567   1234  0  80   0 -     0 exit   0:00 <defunct>
```

Child Process

A child process is the new process created by the parent using fork().
It receives a return value of 0 from the fork() call.

Parent Process

A parent process is the process that creates another process using the fork() system call.
It receives the process ID (PID) of the newly created child.

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

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

    if (pid > 0) {
        // Parent process
        printf("This is Parent Process\n");
        printf("Parent PID: %d\n", getpid());
        printf("Child PID: %d\n", pid);
    }
    else if (pid == 0) {
        // Child process
        printf("This is Child Process\n");
        printf("Child PID: %d\n", getpid());
        printf("Parent PID: %d\n", getppid());
    }

    return 0;
}
```

Output:

```
This is Parent Process
Parent PID: 3120
Child PID: 3121
This is Child Process
Child PID: 3121
Parent PID: 3120
```