

way of the hax0r
n00b to console punk

Chapter 9

Programs That Keep Running

Most commands so far finished in a fraction of a second. Some do not: a download, a build, a server, an editor, or a `sleep` you started by mistake. While one of them holds the terminal, you cannot type the next command, and when it misbehaves you need to stop it without closing the window. The operating system tracks every running program, and the shell gives you handles on the ones it started.

The concepts in this chapter are:

- a **process**, its **process ID** (PID), and its parent;
- listing processes and finding one by name;
- **foreground** and **background** execution, and the shell's **jobs**;
- **signals** and the difference between asking a program to stop and forcing it;
- **exit status** of a program that was stopped;
- inspecting **resources**: CPU, memory, load, and disk.

The examples start and stop harmless programs such as `sleep`. Do not practice stopping on programs you did not start.

Processes

A **process** is a running program: the code, its memory, its open files, and its connection to a terminal if it has one. Starting a command creates a new process, and when the command finishes, the process ends and reports an exit status to its **parent**, the process that started it. Your shell is the parent of everything you run from it, and the shell itself is a child of the terminal application or login manager.

Each process has a number, its **PID**, which is how every other tool refers to it. PIDs are reused after a process ends, so a number from an hour ago may belong to something else now.

List Processes

Bash/zsh:

```
$ ps
$ ps aux
$ pgrep -l sleep
```

`ps` alone lists processes attached to this terminal. `ps aux` lists every process on the system with its user, CPU and memory share, state, and command line; on Linux and macOS the `aux` form works without a hyphen. `pgrep` searches by name and prints matching PIDs, with `-l` adding the name. [1], [2]

PowerShell:

```
$ Get-Process
$ Get-Process -Name pwsh
$ Get-Process | Sort-Object CPU -Descending | Select-Object -First 5 ProcessName, Id, CPU
```

`Get-Process` returns objects with `Id`, `ProcessName`, `CPU`, and `WorkingSet` properties, so sorting and filtering use the pipeline from Chapter 7. `$PID` holds the current PowerShell's own process ID. [3]

Command Prompt:

```
$ tasklist
$ tasklist /fi "imagename eq notepad.exe"
```

`tasklist` lists processes, and `/fi` filters them. [4]

Foreground and Background

A command run normally is in the **foreground**: the shell waits for it, and it owns the terminal's keyboard. A **background** command runs while the shell returns to the prompt. The shell keeps a list of the processes it started, called **jobs**, with a small job number for each.

Bash/zsh

Append `&` to start a command in the background:

```
$ sleep 60 &
[1] 48213
$ jobs
[1]+  Running                  sleep 60 &
```

The shell prints the job number and the PID and returns the prompt. `jobs` lists the shell's jobs; `fg` brings the most recent one to the foreground, and `fg %2` picks job 2. [5], [6]

The other direction works too. Start `sleep 60` in the foreground, press `Ctrl+Z`, and the shell reports `Stopped`. The process is paused, not gone. `bg` resumes it in the background, and `fg` resumes it in the foreground. That is the accidental `Ctrl+Z` from Chapter 4, and `fg` is the way back.

A background job that writes to the terminal still writes over your prompt, so redirect its output to a file. And a background job belongs to the shell that started it: closing that terminal usually ends the job. `nohup` command `&` starts a command that survives the shell's exit, writing its output to `nohup.out`. [7]

PowerShell

PowerShell has **jobs** too, but they run in a separate PowerShell process rather than sharing the terminal:

```
$ $j = Start-Job { Start-Sleep 5; 'finished' }
$ Get-Job
$ Receive-Job $j -Wait
finished
$ Remove-Job $j
```

Start-Job runs a script block in the background and returns a job object. Get-Job lists jobs, Receive-Job collects their output, and Remove-Job cleans up. In PowerShell 7, sleep 5 & at the end of a pipeline does the same as Start-Job. [8], [9]

To start an ordinary program without waiting for it, use Start-Process:

```
$ Start-Process notepad
$ $p = Start-Process sleep -ArgumentList 60 -PassThru
$ $p.Id
```

-PassThru returns the process object so that you can inspect or stop it later. Start-Process is not a job; the program runs on its own, and closing PowerShell does not stop it. [10]

Command Prompt

start launches a program in a new window or, with /b, in the same window without waiting:

```
$ start notepad
$ start /b ping -n 10 localhost
```

Command Prompt has no job list; use tasklist and taskkill to find and stop what start launched. [11]

Stop a Program

Programs stop for three reasons: they finish, they are asked to stop, or they are forced to. Prefer them in that order.

Ctrl+C in the foreground asks the program to stop, and most programs do. For a background or unrelated process, you need its PID or name.

Signals on UNIX

Linux and macOS ask and force with **signals**, small numbered messages the operating system delivers to a process. Two matter every day:

- **TERM** (15) asks the program to terminate. It can clean up first, and it can ignore the request. kill sends this by default.
- **KILL** (9) ends the process immediately. The program gets no chance to save or clean up, and it cannot ignore the signal. [12]

```
$ sleep 60 &
[1] 48301
$ kill 48301
$ jobs
[1]+  Terminated                  sleep 60
$ sleep 60 &
[2] 48310
$ kill -KILL %2
```

kill accepts a PID or a job number with %. kill -KILL and kill -9 mean the same; kill -TERM and kill -15 are the default. pkill sleep sends a signal to every process with a matching name, and killall does the same on systems that have it. [2], [6]

A process ended by a signal reports an exit status of 128 plus the signal number. 143 means TERM and 137 means KILL, which is how a script or a log tells you why something stopped. [13]

Ctrl+C sends INT (2), which programs treat like TERM. Ctrl+Z sends TSTP, the stop behind job control.



Trap: Reach for kill -9 last. A database or editor killed with it can leave a corrupt file or a stale lock. Send TERM, wait a few seconds, check with ps or jobs, and only then force it.

PowerShell

```
$ Stop-Process -Id 48301
$ Stop-Process -Name sleep
$ Get-Process notepad | Stop-Process
```

Stop-Process ends processes by ID, by name, or from the pipeline. On Windows it is forceful: the process is terminated without a request to close. Stop-Process -Confirm asks before each one, and -WhatIf from Chapter 3 works here too. [14]

On Windows, a graphical program can be asked to close politely with (Get-Process notepad).CloseMainWindow(), which is what clicking its close button does.

Command Prompt

```
$ taskkill /pid 48301
$ taskkill /im notepad.exe
$ taskkill /im notepad.exe /f
```

Without /f, taskkill asks a windowed program to close; with /f, it forces termination. /im selects by image name, which is the executable's filename. [15]

Watch the Machine

When everything is slow, the question is which process is using what.

Bash/zsh:

```
$ uptime
$ top
$ free -h
$ df -h
$ du -sh ~/Downloads
```

uptime ends with the **load average**: the number of processes wanting to run, averaged over one, five, and fifteen minutes. A load near the number of CPU cores, which nproc prints on Linux, means the machine is busy. top shows a live, sorted list of processes; press q to leave it, and read its help with ?. htop, where installed, shows the same with a friendlier display. [16], [17]

free -h reports memory on Linux; macOS uses vm_stat or Activity Monitor instead. df -h shows disk space by filesystem, and du -sh measures one directory tree. [18], [19], [20]

time command reports how long a command took, and timeout 10 command stops it if it runs longer than ten seconds. [21], [22]

PowerShell:

```
$ Get-Process | Sort-Object WorkingSet -Descending | Select-Object -First 5 ProcessName, WorkingSet
$ Get-PSDrive -PSProvider FileSystem
$ Measure-Command { Get-ChildItem -Recurse }
```

WorkingSet is memory in bytes; CPU is seconds of processor time used so far. Get-PSDrive shows used and free space per drive, and Measure-Command times a script block. On Windows, Task Manager and Get-Counter give live numbers; on Linux and macOS, use that system's top. [3], [23]

Command Prompt:

```
$ tasklist /v
$ systeminfo
```

tasklist /v adds memory and CPU time to the listing, and systeminfo reports totals. For live monitoring, open Task Manager or use PowerShell. [4]

Key Points

- A process is a running program with a PID; ps, Get-Process, and tasklist list them.
- & runs a command in the background in Bash and zsh; jobs, fg, bg, and Ctrl+Z manage the shell's jobs.
- PowerShell jobs run in a separate process with Start-Job, Receive-Job, and Remove-Job; Start-Process launches a detached program.
- UNIX stops programs with signals: TERM asks, KILL forces, and an exit status of 128 plus the signal number tells you which was used.
- kill, Stop-Process, and taskkill end processes by PID or name; try the polite form before the forceful one.
- top, free, df, and uptime report resource use on UNIX; Get-Process sorted by CPU or WorkingSet and Get-PSDrive do it in PowerShell.

Level Up

n00bs finish the chapter here. You can find a process, move it between foreground and background, stop it, and see what the machine is doing. Hackers look at what a process is doing from the outside.

Explore `/proc` on Linux: `cat /proc/PID/status` and `ls -l /proc/PID/fd` show a process's state and open files. Compare with `ls -l /proc/PID/fd` on macOS and `Get-Process -Id PID | Select-Object *` in PowerShell.

Try `strace -p PID` on Linux or `dtruss` on macOS to watch the system calls a program makes. Explain what a program blocked on read is waiting for.

Read about **services**: `systemctl` on Linux, `launchctl` on macOS, and `Get-Service` on Windows. How does a service differ from a background job you started in a shell?

Investigate `nice` and `renice`, and PowerShell's `PriorityClass` property, then explain what priority changes and what it does not. Then read about **cgroups** and explain how a container limits what a process can use.

Exercises

1. **What does this do?** Predict the output of `sleep 30 & jobs; kill %1; sleep 1; jobs`, then run it.
2. **What does this do?** Predict what `$p = Start-Process sleep -ArgumentList 5 -PassThru; $p.HasExited; Start-Sleep 6; $p.HasExited` prints in PowerShell.
3. **What is the problem?** A user runs a server with `python server.py &`, closes the terminal, and the server is gone. Explain why and give two ways to keep it running.
4. **What is the problem?** After `Ctrl+Z` in Vim, the terminal shows a prompt and Vim seems closed, but opening the same file again warns about a swap file. Explain what happened and how to get back.
5. **Why is it like this?** A program stopped with `kill -9` reports exit status 137. Explain the arithmetic and why the shell reports it that way.
6. **Why is it like this?** `Stop-Process` on Windows does not give a program the chance to save its work, while `taskkill` without `/f` does. Explain what each one sends.
7. **How do you** find the process using the most memory, in Bash on Linux and in PowerShell?
8. **How do you** stop a program that ignores `Ctrl+C`, from another terminal tab?

References

- [1] procps-ng Project, “Ps(1): Report a snapshot of the current processes.” Available: <https://gitlab.com/procps-ng/procps/-/blob/master/man/ps.1>
- [2] procps-ng Project, “Pgrep(1), pkill(1): Look up or signal processes based on name.” Available: <https://man7.org/linux/man-pages/man1/pgrep.1.html>
- [3] Microsoft, “Get-process.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-process>
- [4] Microsoft, “Windows commands: tasklist.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/tasklist>
- [5] GNU Project, “Bash reference manual: Job control basics.” Available: https://www.gnu.org/software/bash/manual/html_node/Job-Control-Basics.html
- [6] GNU Project, “Bash reference manual: Job control builtins.” Available: https://www.gnu.org/software/bash/manual/html_node/Job-Control-Builtins.html
- [7] GNU Project, “GNU coreutils manual: Nohup: Run a command immune to hangups.” Available: https://www.gnu.org/software/coreutils/manual/html_node/nohup-invocation.html
- [8] Microsoft, “About_jobs.” Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_jobs
- [9] Microsoft, “Start-job.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/start-job>
- [10] Microsoft, “Start-process.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/start-process>
- [11] Microsoft, “Windows commands: start.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/start>
- [12] Linux man-pages Project, “Signal(7): Overview of signals.” Available: <https://man7.org/linux/man-pages/man7/signal.7.html>
- [13] GNU Project, “Bash reference manual: Exit status.” Available: https://www.gnu.org/software/bash/manual/html_node/Exit-Status.html
- [14] Microsoft, “Stop-process.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.management/stop-process>
- [15] Microsoft, “Windows commands: taskkill.” Available: <https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/taskkill>
- [16] procps-ng Project, “Top(1): Display linux processes.” Available: <https://man7.org/linux/man-pages/man1/top.1.html>

- [17] procps-ng Project, “Uptime(1): Tell how long the system has been running.” Available: <https://man7.org/linux/man-pages/man1/uptime.1.html>
- [18] procps-ng Project, “Free(1): Display amount of free and used memory.” Available: <https://man7.org/linux/man-pages/man1/free.1.html>
- [19] GNU Project, “GNU coreutils manual: Df: Report file system space usage.” Available: https://www.gnu.org/software/coreutils/manual/html_node/df-invocation.html
- [20] GNU Project, “GNU coreutils manual: Du: Estimate file space usage.” Available: https://www.gnu.org/software/coreutils/manual/html_node/du-invocation.html
- [21] GNU Project, “Bash reference manual: pipelines.” Available: https://www.gnu.org/software/bash/manual/html_node/Pipelines.html
- [22] GNU Project, “GNU coreutils manual: Timeout: Run a command with a time limit.” Available: https://www.gnu.org/software/coreutils/manual/html_node/timeout-invocation.html
- [23] Microsoft, “Measure-command.” Available: <https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/measure-command>