Introduction to Operating Systems

3. Processes

2026-11-03

USO Team

Bibliography

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 1 - *The Big Picture*
     - Section 1.3.1 - *Process Management*
   - Chapter 2 - *Basic Commands and Directory Hierarchy*
     - Section 2.14 - *Shell Input and Output*
     - Section 2.16 - *Listing and Manipulating Processes*
     - Section 2.17.2 - *Working with Symbolic Links*
   - Chapter 3 - *Devices*
     - Section 3.1 - *Device Files*
   - Chapter 4 - *Disks and Filesystems*
     - Section 4.2.12 - *Special-Purpose Filesystems*
   - Chapter 6 - *How User Space Starts*
     - Sections 6.1 - 6.6
   - Chapter 8 - *A Closer Look at Processes and Resource Utilization*
     - Sections 8.1 - 8.3
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 2 - *Utilizarea sistemului de fișiere*
     - Section 2.3.4.1 - *Crearea legăturilor simbolice*
     - Section 2.4 - *Redirectarea intrării sau ieșirii*
   - Chapter 3 - *Pachete software*
     - Sections 3.1 - 3.3
   - Chapter 4 - *Procese*
     - Sections 4.1 - 4.5, 4.8 - 4.9
   - Chapter 8 - *Componente hardware*
     - Section 8.6 - *Abstractizarea dispozitivelor în Linux*
   - Chapter 9 - *Pornirea sistemului*
     - Section 9.6 - *Pornirea init și a serviciilor de startup în Linux*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

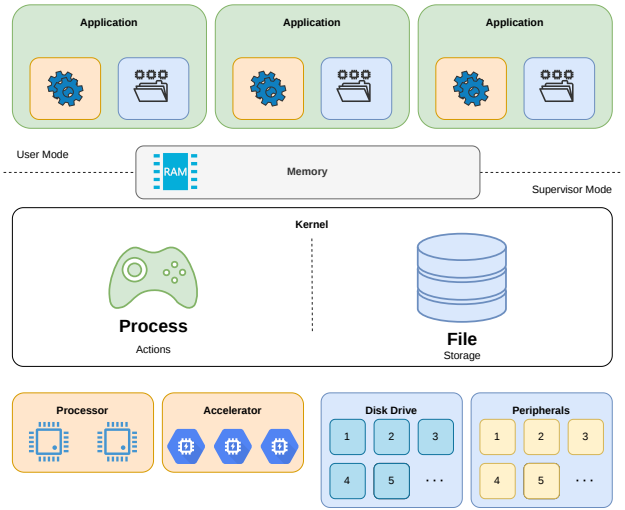
Operating System: abstractions

Actions

- *Applications*
- use the *Processor* and *Accelerators* (GPU, Neural Engine, etc)

Data

- everything is a file
- peripherals are viewed as files (*POSIX*)
  - `/dev/input/keyboard` - keyboard
  - `/dev/fb` - screen (framebuffer)
  - `/dev/sda` - Disk Drive A (the first drive)



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Installing Applications

App repositories

Bibliography

For this section

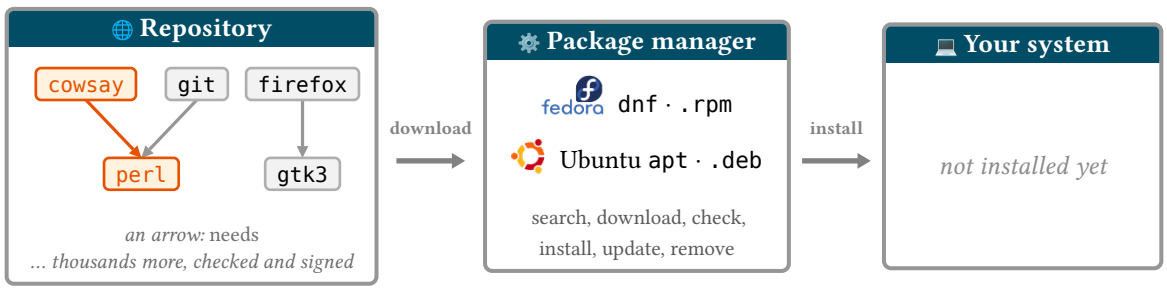
1. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 3 - *Pachete software*
     - Section 3.1 - *Pachete software*
     - Section 3.2 - *Gestiunea pachetelor*
     - Section 3.3 - *Gestiunea pachetelor în Linux*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

App Repositories

The Linux app store, long before the app stores

- applications come as **packages**: the program, its files and the list of packages it needs (*dependencies*)

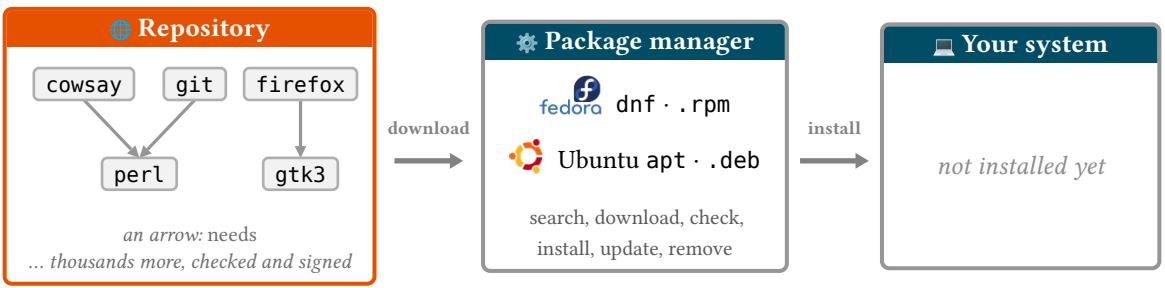


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

App Repositories

The Linux app store, long before the app stores

- applications come as **packages**: the program, its files and the list of packages it needs (*dependencies*)
- the distribution keeps them in a **repository**: a server with thousands of checked and signed packages

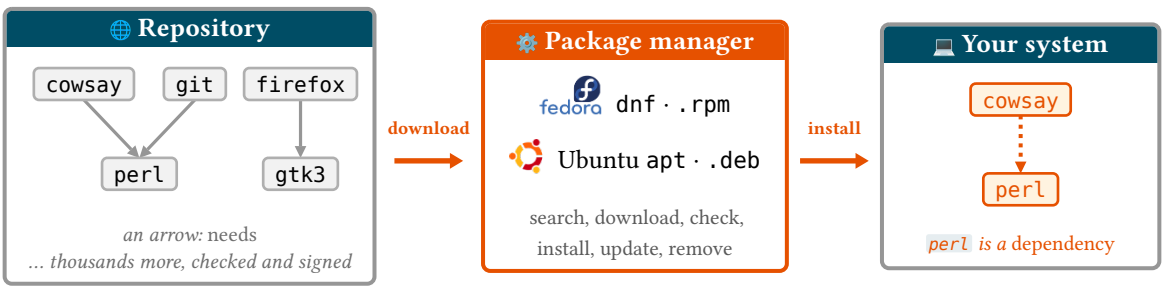


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

App Repositories

The Linux app store, long before the app stores

- applications come as **packages**: the program, its files and the list of packages it needs (*dependencies*)
- the distribution keeps them in a **repository**: a server with thousands of checked and signed packages
- the **package manager** searches, downloads, installs, updates and removes them, dependencies included



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Fedora: `dnf`

Search and install `cowsay`

```
$ dnf search cowsay
Matched fields: name (exact)
 cowsay.noarch     Configurable speaking/thinking cow
Matched fields: name
 xcowsay.x86_64    Displays a cute cow and message on your desktop
```

💡 `dnf search` looks for the name in the repositories: `cowsay` is in the `fedora` repository

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Fedora: `dnf`

Search and install `cowsay`

```
$ dnf search cowsay
Matched fields: name (exact)
 cowsay.noarch     Configurable speaking/thinking cow
Matched fields: name
 xcowsay.x86_64    Displays a cute cow and message on your desktop
$ sudo dnf install cowsay
Package Arch   Version        Repository      Size
Installing:
 cowsay noarch 0:3.8.4-5.fc44 fedora      80.5 KiB
Is this ok [y/N]: y
```

💡 installing changes the system for all the users: it needs `sudo` (the *administrator*), and asks before it starts

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Ubuntu: `apt`



Search and install `cowsay`

```
$ apt search cowsay
cowsay/noble 3.03+dfsg2-8 all
  configurable talking cow
xcowsay/noble 1.6-1build2 amd64
  Graphical configurable talking cow
```

💡 the same idea, another package manager: `apt search` looks in the Ubuntu repositories (`noble` is Ubuntu 24.04)

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Ubuntu: `apt`



Search and install `cowsay`

```
$ apt search cowsay
cowsay/noble 3.03+dfsg2-8 all
  configurable talking cow
xcowsay/noble 1.6-1build2 amd64
  Graphical configurable talking cow
$ sudo apt install cowsay
The following NEW packages will be installed:
  cowsay libgdbm-compat4t64 libgdbm6t64 libperl5.38t64
  libtext-charwidth-perl netbase perl perl-modules-5.38
Do you want to continue? [Y/n] y
```

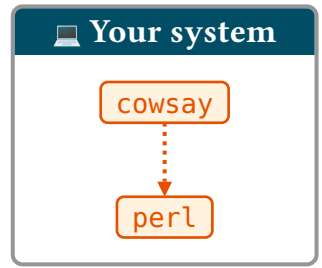
💡 `cowsay` is written in Perl: `apt` installs the **dependencies** too, `perl` and its libraries

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Dependencies Are Packages Too

After installing `cowsay`, `perl` is on the system

```
$ which cowsay perl
/usr/games/cowsay
/usr/bin/perl
```



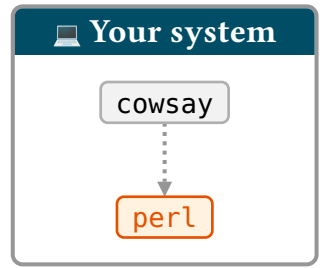
💡 `apt` installed both: `cowsay`, and `perl` because `cowsay` needs it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Dependencies Are Packages Too

After installing `cowsay`, `perl` is on the system

```
$ which cowsay perl
/usr/games/cowsay
/usr/bin/perl
$ perl -e 'print "hello from perl\n"'
hello from perl
```



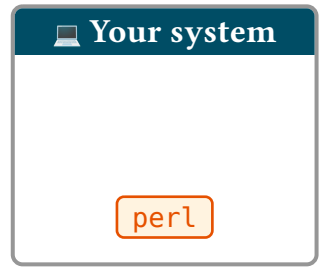
💡 `perl` is a normal package: you, or any other program, can use it **by itself**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Dependencies Are Packages Too

After installing `cowsay`, `perl` is on the system

```
$ which cowsay perl
/usr/games/cowsay
/usr/bin/perl
$ perl -e 'print "hello from perl\n"'
hello from perl
$ sudo apt remove cowsay
The following packages will be REMOVED:
  cowsay
```



💡 removing `cowsay` does not remove `perl`; `sudo apt autoremove` removes the dependencies that nothing needs anymore

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Package Managers Cheat Sheet

Fedora and Ubuntu

	fedora	Ubuntu
search	`dnf search NAME`	`apt search NAME`
details	`dnf info NAME`	`apt show NAME`
install	`sudo dnf install NAME`	`sudo apt install NAME`
remove	`sudo dnf remove NAME`	`sudo apt remove NAME`
remove unused dependencies	`sudo dnf autoremove`	`sudo apt autoremove`
update everything	`sudo dnf upgrade`	`sudo apt update` `sudo apt upgrade`

💡 `apt update` only refreshes the list of packages, `apt upgrade` installs the new versions; `dnf` refreshes the list by itself

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Process

From a file to an action

Bibliography

For this section

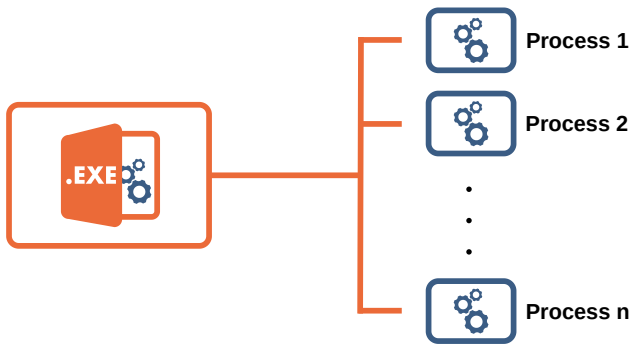
1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 1 - *The Big Picture*
     - Section 1.3.1 - *Process Management*
   - Chapter 6 - *How User Space Starts*
     - Sections 6.2 - 6.3 - *Identifying Your init, systemd*
   - Chapter 8 - *A Closer Look at Processes and Resource Utilization*
     - Section 8.1 - *Tracking Processes*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 4 - *Procese*
     - Sections 4.1 - 4.3 - *Programe și procese, Resursele și atributele unui proces, Ierarhia de procese*
     - Section 4.9 - *Anexa: Internele pornirii unui proces. Loading*
   - Chapter 9 - *Pornirea sistemului*
     - Section 9.6 - *Pornirea init și a serviciilor de startup în Linux*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Executable vs Process

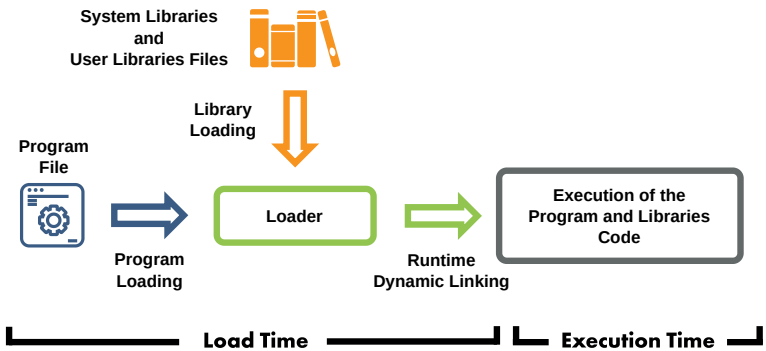
Files to *actions*

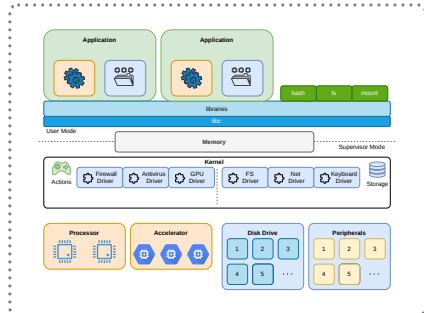
📁 one executable file ➡️ multiple processes ⚙️⚙️...⚙️



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Loading an Executable

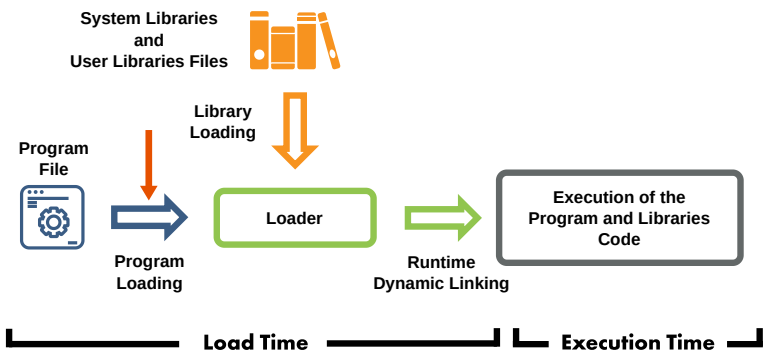


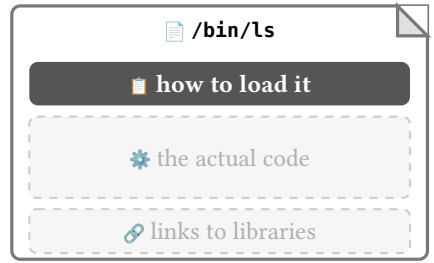


💡 the **loader** turns the program file into a running process, in a few steps

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Loading an Executable



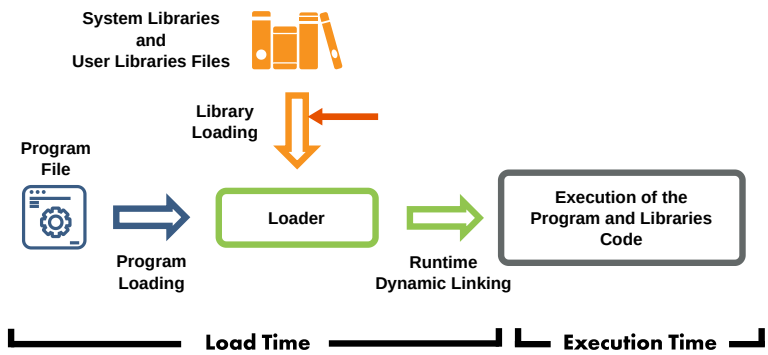


Executable File

💡 **program loading**: the loader reads the executable and copies its code and data into memory

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Loading an Executable



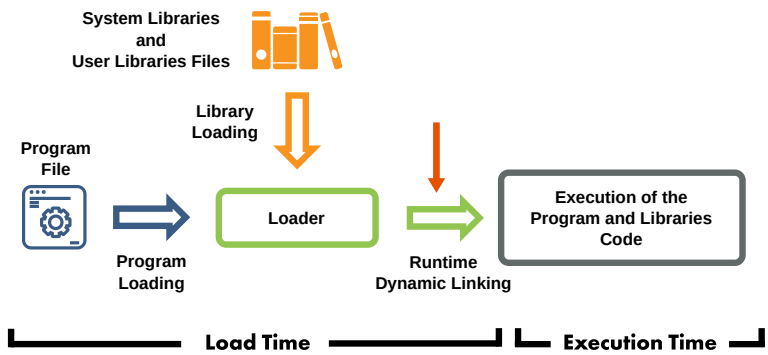


Executable File

💡 **library loading**: it finds the libraries the program needs (`libc.so.6`, ...) and loads them too

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Loading an Executable



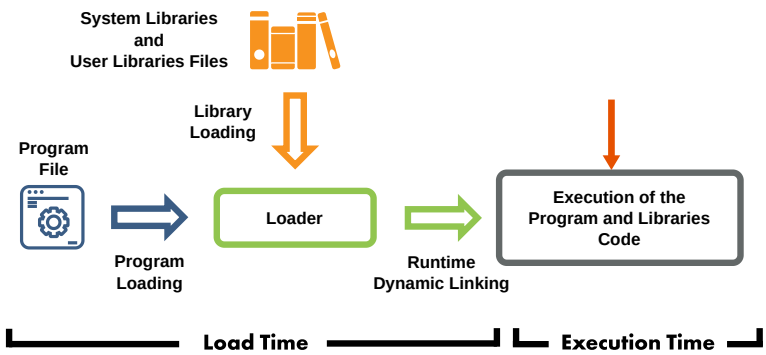


Executable File

💡 **runtime dynamic linking**: it connects the program's calls to the functions inside the libraries

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Loading an Executable





Executable File

💡 **execution**: the processor jumps to the start of the program, the code of the program and of the libraries runs

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Process' Resources

Identification

- *PID*: **P**rocess **ID**
- *PPID*: **P**arent **P**rocess **ID**

⚙️ Actions

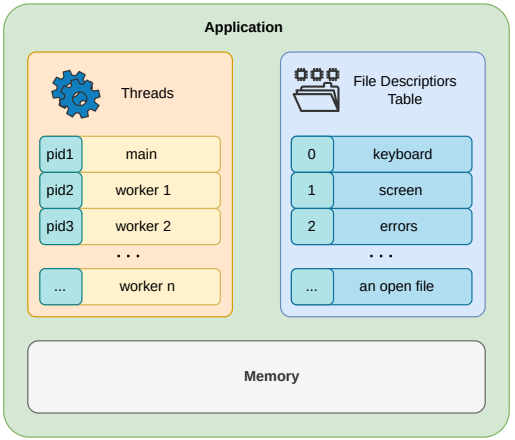
- one or multiple threads executing in parallel

💾 Data

- access to files
  - actual files
  - peripheral files (`/dev/...`)

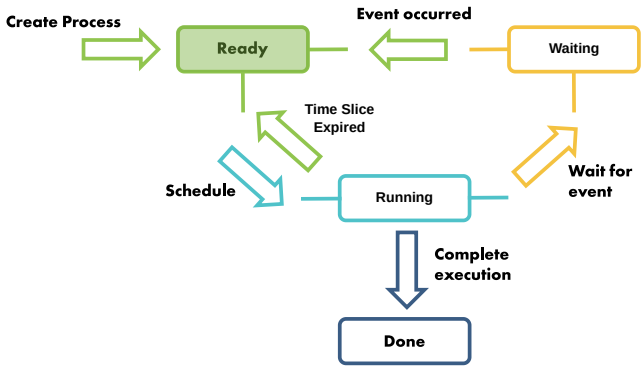
🧠 Memory

- code
- variables



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Process States

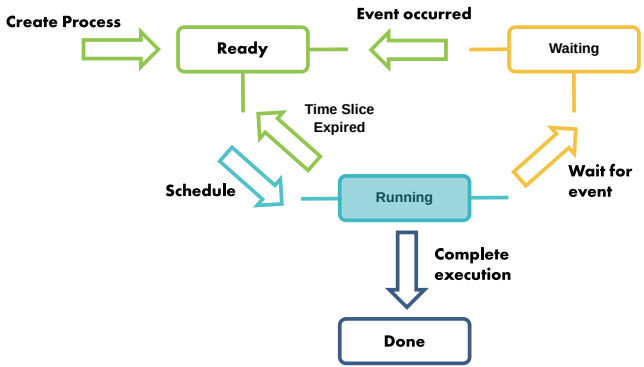



```
$ ps -e -o pid,stat,comm
    PID STAT COMMAND
      1 Ss   systemd
    ...
   4100 Ss   bash
   4210 S    sleep
   4211 R    yes
   4213 Sl   myapp
   4214 Z    myapp-worker <defunct>
   4220 R+   ps
```

Ready - has everything it needs, waits for a free **processor**
in `ps` it is `R`, like **Running**: `ps` cannot tell them apart

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Process States



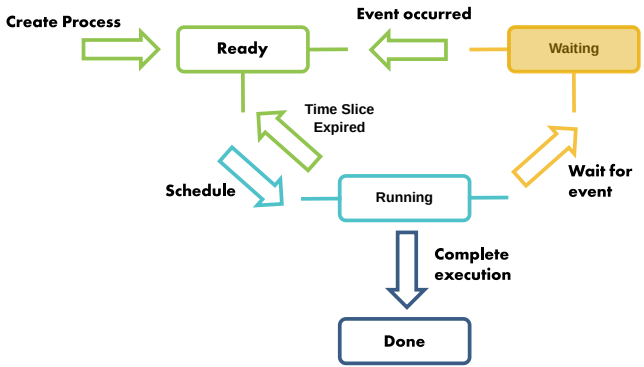

```
$ ps -e -o pid,stat,comm
    PID STAT COMMAND
      1 Ss   systemd
    ...
   4100 Ss   bash
   4210 S    sleep
   4211 R    yes
   4213 Sl   myapp
   4214 Z    myapp-worker <defunct>
   4220 R+   ps
```

Running - its code runs on a processor; when its *time slice* expires, it goes back to **Ready**

`R` in `ps`: `yes` computes all the time, `ps` runs while it makes the list

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Process States



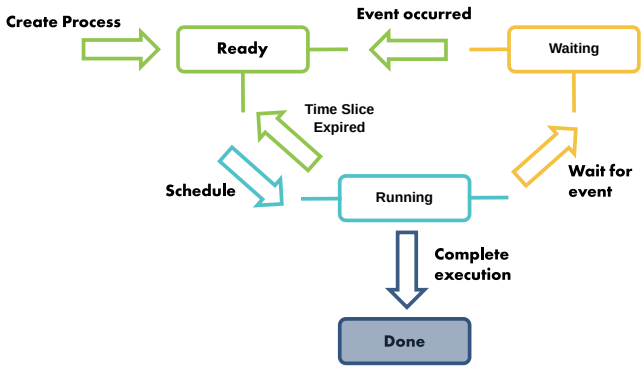

```
$ ps -e -o pid,stat,comm
    PID STAT COMMAND
      1 Ss   systemd
    ...
   4100 Ss   bash
   4210 S    sleep
   4211 R    yes
   4213 Sl   myapp
   4214 Z    myapp-worker <defunct>
   4220 R+   ps
```

Waiting - waits for an **event**: data from the disk, a key press, a network packet; it does not use the processor

`S` (*sleeping*) in `ps`: `bash` waits for you to type, `sleep` waits for its timer

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Process States




```
$ ps -e -o pid,stat,comm
    PID STAT COMMAND
      1 Ss   systemd
    ...
   4100 Ss   bash
   4210 S    sleep
   4211 R    yes
   4213 Sl   myapp
   4214 Z    myapp-worker <defunct>
   4220 R+   ps
```

Done - it has finished (`exit`), it waits for the parent to read its *exit code*

`Z` (*zombie*[1]) in `ps`: `myapp-worker` has finished, but its parent, `myapp`, has not read its exit code yet

[1] *zombie* processes are explained later, in *Zombie Process*

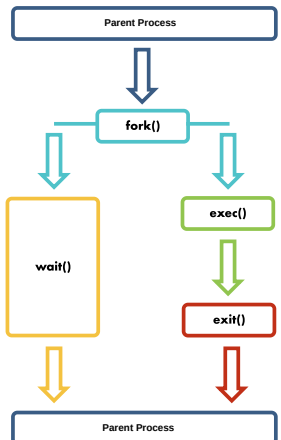
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Creating a Process

Forking a parent and loading an executable

Child

Parent



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

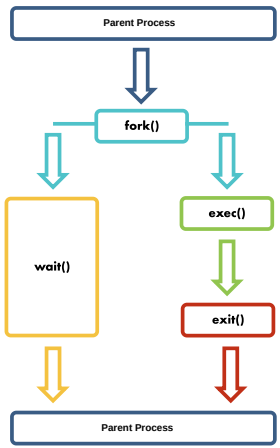
Creating a Process

Forking a parent and loading an executable

The parent process calls `fork` - *makes a **clone** of the parent*

Child

Parent



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Creating a Process

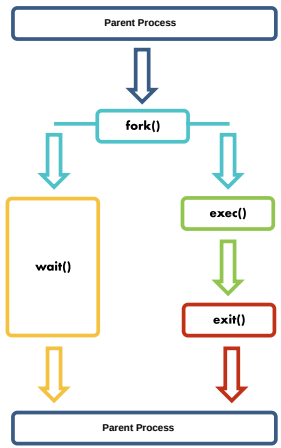
Forking a parent and loading an executable

The parent process calls `fork` - *makes a **clone** of the parent*

from now on we have **two identical processes**

Child

Parent



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Creating a Process

Forking a parent and loading an executable

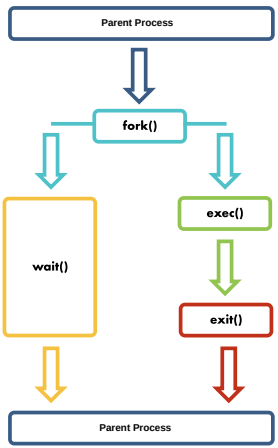
The parent process calls `fork` - *makes a **clone** of the parent*

from now on we have **two identical processes**

Child

1. Calls `exec` - *replaces the executable*

Parent



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Creating a Process

Forking a parent and loading an executable

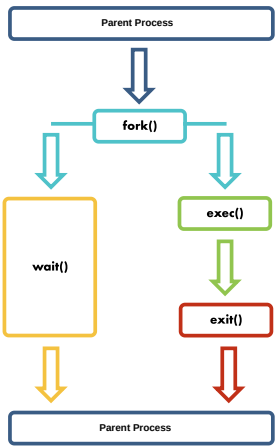
The parent process calls `fork` - *makes a **clone** of the parent*

from now on we have **two identical processes**

Child

1. Calls `exec` - *replaces the executable*
2. Eventually calls `exit` (or `return` in `main`) and returns an *exit code*

Parent



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Creating a Process

Forking a parent and loading an executable

The parent process calls `fork` - *makes a **clone** of the parent*

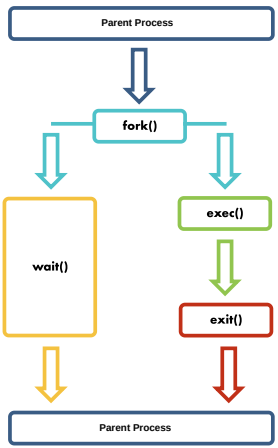
from now on we have **two identical processes**

Child

1. Calls `exec` - *replaces the executable*
2. Eventually calls `exit` (or `return` in `main`) and returns an *exit code*

Parent

1. Does other work or *may wait* for the child process



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Creating a Process

Forking a parent and loading an executable

The parent process calls `fork` - *makes a **clone** of the parent*

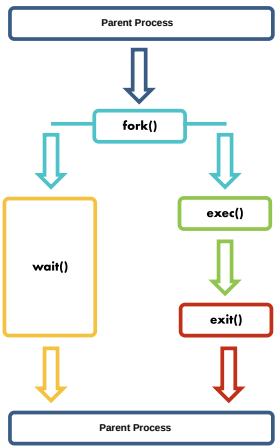
from now on we have **two identical processes**

Child

1. Calls `exec` - *replaces the executable*
2. Eventually calls `exit` (or `return` in `main`) and returns an *exit code*

Parent

1. Does other work or *may wait* for the child process
2. Eventually calls `wait` to read the *exit code* of the child



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
|   # The kernel's first process (part of the kernel itself)
|   # Manages CPU scheduling and idling
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
    # The kernel's first process (part of the kernel itself)
    # Manages CPU scheduling and idling
    |
    └── [PID 1]  systemd (called init)
          # The first user-space process started by the kernel
          # Responsible for initializing the rest of the system
          # Mounts filesystems, starts targets, and manages services
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
   # The kernel's first process (part of the kernel itself)
   # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
      # Internal kernel helpers for scheduling, I/O, and interrupts
      # These run in the background as part of the kernel itself
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
    # The kernel's first process (part of the kernel itself)
    # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
│      ├── [PID 3+]  systemd-udevd
│      │        # Handles dynamic device detection (e.g., USB, disks)
│
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
    # The kernel's first process (part of the kernel itself)
    # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
│      ├── [PID 3+]  systemd-udevd
│      ├── [PID 5+]  NetworkManager (or systemd-networkd)
│      │       # Configures and manages network interfaces
│
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
   # The kernel's first process (part of the kernel itself)
   # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
│      ├── [PID 3+]  systemd-udevd
│      ├── [PID 5+]  NetworkManager (or systemd-networkd)
│      ├── [PID 8+]  gdm / sddm / lightdm (optional)
│      │       # Display manager - starts graphical login session
│
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
    # The kernel's first process (part of the kernel itself)
    # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
│     ├── [PID 3+]  systemd-udevd
│     ├── [PID 5+]  NetworkManager (or systemd-networkd)
│     ├── [PID 8+]  gdm / sddm / lightdm (optional)
│     └── [PID 10+]  user@1000.service
│           # Per-user systemd instance
│           # Manages user-level services after login
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
    # The kernel's first process (part of the kernel itself)
    # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
│      ├── [PID 3+]  systemd-udevd
│      ├── [PID 5+]  NetworkManager (or systemd-networkd)
│      ├── [PID 8+]  gdm / sddm / lightdm (optional)
│      └── [PID 10+]  user@1000.service
│             ├── [PID 11+]  bash / zsh / fish
│             │       # User's shell process after login
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
    # The kernel's first process (part of the kernel itself)
    # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
│     ├── [PID 3+]  systemd-udevd
│     ├── [PID 5+]  NetworkManager (or systemd-networkd)
│     ├── [PID 8+]  gdm / sddm / lightdm (optional)
│     └── [PID 10+]  user@1000.service
│           ├── [PID 11+]  bash / zsh / fish
│           ├── [PID 12+]  Xorg / Wayland
│           │       # Graphical display server (for GUI sessions)
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
   # The kernel's first process (part of the kernel itself)
   # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
│     ├── [PID 3+]  systemd-udevd
│     ├── [PID 5+]  NetworkManager (or systemd-networkd)
│     ├── [PID 8+]  gdm / sddm / lightdm (optional)
│     └── [PID 10+]  user@1000.service
│           ├── [PID 11+]  bash / zsh / fish
│           ├── [PID 12+]  Xorg / Wayland
│           └── [PID 13+]  gnome-shell / plasma / sway
│                    # User's desktop environment or window manager
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How Linux Starts

The first process

```
[kernel]  interrupt handler / scheduler
    # The kernel's first process (part of the kernel itself)
    # Manages CPU scheduling and idling
│
├── [PID 1]  systemd (called init)
│     ├── [PID 3+]  systemd-udevd
│     ├── [PID 5+]  NetworkManager (or systemd-networkd)
│     ├── [PID 8+]  gdm / sddm / lightdm (optional)
│     └── [PID 10+]  user@1000.service
│           ├── [PID 11+]  bash / zsh / fish
│           ├── [PID 12+]  Xorg / Wayland
│           └── [PID 13+]  gnome-shell / plasma / sway
│                    # User's desktop environment or window manager
└── [PID ...]  kernel threads (kthreadd, kworker, ksoftirqd, etc.)
```

⚠ **`init` cannot stop!**

or the whole system will panic!

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

😕 Orphan Process

What happens when the parent process ends

- child processes remain without a parent
- orphan processes are *reparented* to `init` (PID 1)

```
[PID 1]  systemd (init)
     # PID 1: init, adopts orphaned processes
      │
      ├── [PID 10+]  user@1000.service
      │      │
      │      ├── [PID 100]  myapp
      │      │      # Application master process
      │      │      │
      │      │      ├── [PID 200]  myapp-worker
      │      │      │      # Worker doing background jobs
      │      │      └── [PID 201]  myapp-helper
      │      │             # Helper process
...
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

😕 Orphan Process

What happens when the parent process ends

- child processes remain without a parent
- orphan processes are *reparented* to `init` (PID 1)

```
[PID 1]  systemd (init)
│     # PID 1: init, adopts orphaned processes
│     │
│     ├── [PID 10+]  user@1000.service
│     ...
├── [PID 200]  myapp-worker    <-- reparented
│     # Was child of PID 100; now adopted by PID 1
├── [PID 201]  myapp-helper    <-- reparented
│     # Also adopted by PID 1
...
```

💡 `myapp` (PID 100) has ended

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

😕 Orphan Process

What happens when the parent process ends

- child processes remain without a parent
- orphan processes are *reparented* to `init` (PID 1)

```
[PID 1]  systemd (init)
    # PID 1: init, adopts orphaned processes
    ├── [PID 10+]  user@1000.service
    ...
├── [PID 200]  myapp-worker    <-- reparented
    # Was child of PID 100; now adopted by PID 1
├── [PID 201]  myapp-helper    <-- reparented
    # Also adopted by PID 1
...
```

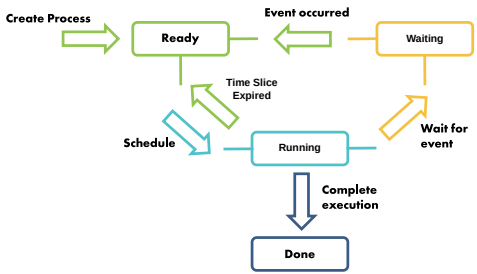
⚠️ `init` has to be present to be able to reparent processes

or the whole system will panic!

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Zombie Process

And this is why `init` cannot stop

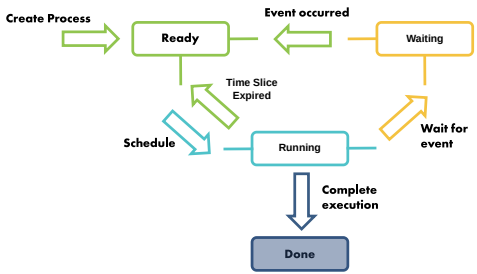


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

🧟 Zombie Process

And this is why `init` cannot stop

- the process will stay in `Done` and will **wait for the parent** to *read its return code* (`wait`)

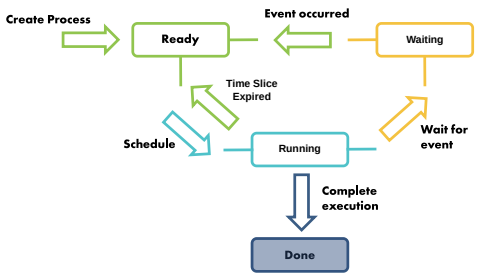


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

🧟 Zombie Process

And this is why `init` cannot stop

- the process will stay in `Done` and will **wait for the parent** to *read its return code* (`wait`)
- the system will **keep all the process' resources allocated** while in `Done`

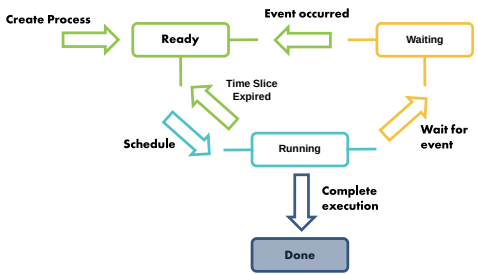


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

🧟 Zombie Process

And this is why `init` cannot stop

- the process will stay in `Done` and will **wait for the parent** to *read its return code* (`wait`)
- the system will **keep all the process' resources allocated** while in `Done`



- until the parent reads the return code of the child, **resources** stay **allocated**, **but not used**: 🧟 zombie

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

😕 Orphan and 🧟 Zombie

Uses up resources

⚠️ `init` has to be **present** to be able to reparent processes

or the whole system will be full of 🧟 zombies

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0



*Linux does not like **permanent** zombies, so it just panics when `init` stops.*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

File Descriptors

How processes access data

Bibliography

For this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 8 - *A Closer Look at Processes and Resource Utilization*
     - Section 8.2 - *Finding Open Files with lsof*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 4 - *Procese*
     - Section 4.4.1 - *Descriptori de fișiere*
     - Section 4.8 - *Anexa: Sistemul de fișiere procfs*

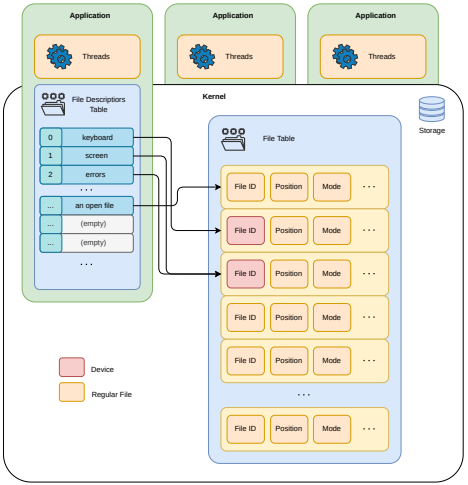
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

File Descriptor Table

How processes access data

- each process has a *File Descriptor Table*
- open files show up in the table
- the **index** in the table is the **file descriptor**

Special Descriptors



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

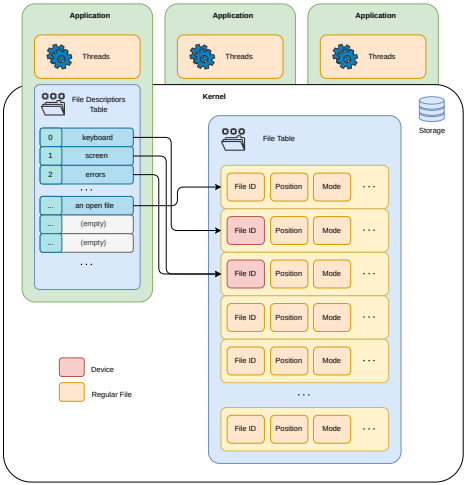
File Descriptor Table

How processes access data

- each process has a *File Descriptor Table*
- open files show up in the table
- the **index** in the table is the **file descriptor**

Special Descriptors

- `0` - *keyboard* - `stdin`



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

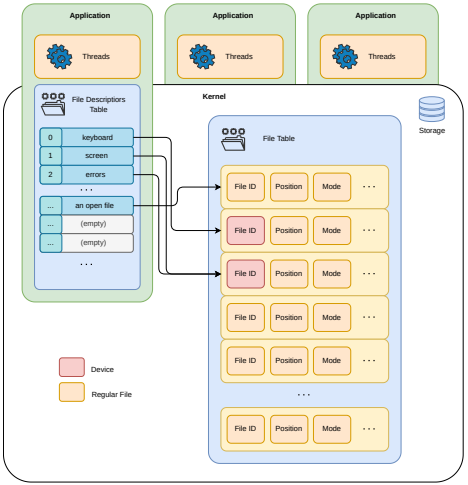
File Descriptor Table

How processes access data

- each process has a *File Descriptor Table*
- open files show up in the table
- the **index** in the table is the **file descriptor**

Special Descriptors

- `0` - *keyboard* - `stdin`
- `1` - *display* - `stdout`



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

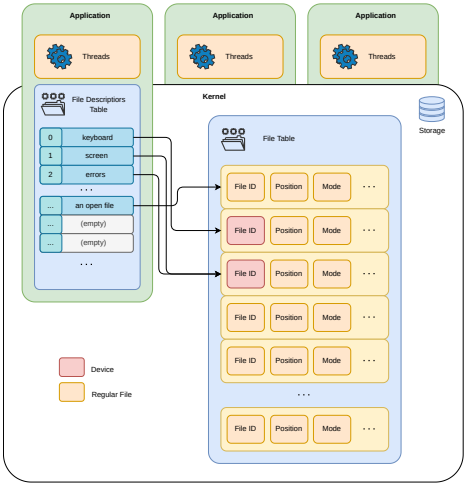
File Descriptor Table

How processes access data

- each process has a *File Descriptor Table*
- open files show up in the table
- the **index** in the table is the **file descriptor**

Special Descriptors

- `0` - *keyboard* - `stdin`
- `1` - *display* - `stdout`
- `2` - *error display* - `stderr`



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`sort` reads a file

The process opens the file by itself

```
$ cat fruits.txt
cherry
apple
banana
$ sort fruits.txt    # the file name is an argument
```

FD Table of `sort`	
0	⌨️ keyboard
1	🖥️ display
2	🖥️ display



💡 the **shell** starts `sort` with `0`, `1` and `2`, and gives it `fruits.txt` as an *argument*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`sort` reads a file

The process opens the file by itself

```
$ cat fruits.txt
cherry
apple
banana
$ sort fruits.txt    # the file name is an argument
```

FD Table of `sort`	
0	⌨️ keyboard
1	🖥️ display
2	🖥️ display
3	📄 `fruits.txt`



💡 `sort` calls `open("fruits.txt")`, the kernel places it at the first free index: `3`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`sort` reads a file

The process opens the file by itself

```
$ cat fruits.txt
cherry
apple
banana
$ sort fruits.txt    # the file name is an argument
apple
banana
cherry
```

FD Table of `sort`	
0	⌨ keyboard
1	🖥 display
2	🖥 display
3	📄 `fruits.txt`



💡 `sort` reads the lines from `3`, sorts them and writes them to `1`, the keyboard (`0`) is not used

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects

Change where a process reads and writes

Bibliography

For this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 2 - *Basic Commands and Directory Hierarchy*
     - Section 2.14 - *Shell Input and Output*
   - Chapter 3 - *Devices*
     - Section 3.1 - *Device Files*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 2 - *Utilizarea sistemului de fișiere*
     - Section 2.4 - *Redirectarea intrării sau ieșirii*
   - Chapter 4 - *Procese*
     - Section 4.4.2 - *Redirectarea în/din fișiere*
     - Sections 4.5.3 - 4.5.4 - *Înlănțuirea comenzilor*, *Comunicarea prin pipe-uri*
   - Chapter 8 - *Componente hardware*
     - Section 8.6 - *Abstractizarea dispozitivelor în Linux*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`<` - redirect `stdin`

The file **replaces** the keyboard as `stdin`

```
$ sort                  # reads the keyboard, until Ctrl+D
```

FD Table of `sort`	
0	⌨ keyboard
1	🖥 display
2	🖥 display



💡 `sort` reads its lines from `0` (`stdin`), which is the keyboard

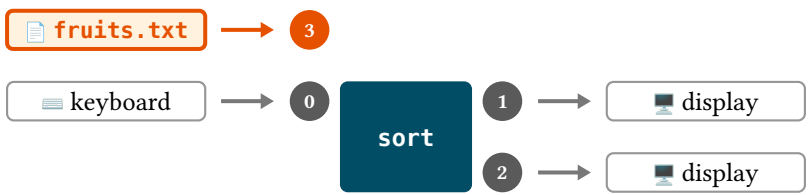
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`<` - redirect `stdin`

The file **replaces** the keyboard as `stdin`

```
$ sort               # reads the keyboard, until Ctrl+D
$ sort < fruits.txt  # reads fruits.txt
```

FD Table of `sort`	
0	⌨️ keyboard
1	🖥️ display
2	🖥️ display
3	📄 `fruits.txt`



💡 the **shell** opens `fruits.txt` for `sort`, it gets the first free index: `3`

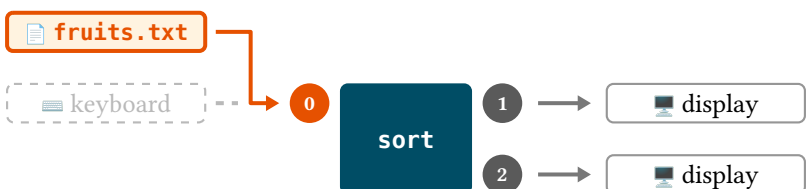
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`<` - redirect `stdin`

The file **replaces** the keyboard as `stdin`

```
$ sort               # reads the keyboard, until Ctrl+D
$ sort < fruits.txt  # reads fruits.txt
apple
banana
cherry
```

FD Table of `sort`	
0	📄 `fruits.txt`
1	🖥️ display
2	🖥️ display



💡 `<` **replaces** `stdin`: `0` is connected to `fruits.txt` instead of the keyboard, `3` is not needed anymore

💡 `sort` does not know that it reads a file, it still just reads from `0` (`stdin`)

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`>` - redirect `stdout`

The process writes into a file instead of the display

```
$ ls                    # writes on the display
Movies  notes.txt  todo.txt
```

FD Table of `ls`	
0	⌨️ keyboard
1	🖥️ display
2	🖥️ display

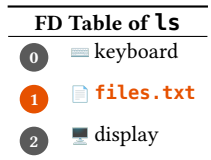


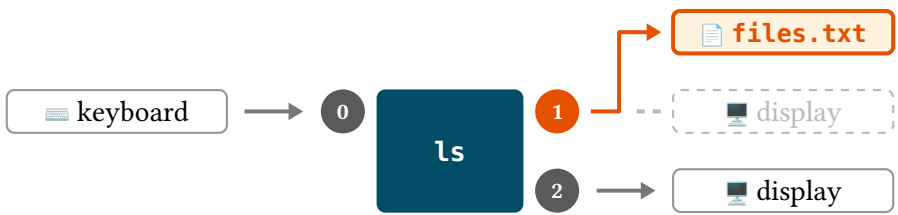
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`>` - redirect `stdout`

The process writes into a file instead of the display

```
$ ls                    # writes on the display
Movies  notes.txt  todo.txt
$ ls > files.txt        # nothing shows up
```





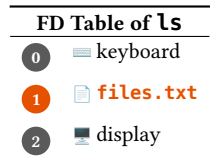
⚠️ `>` *empties* `files.txt` first (or creates it), use `>>` to **append** to it

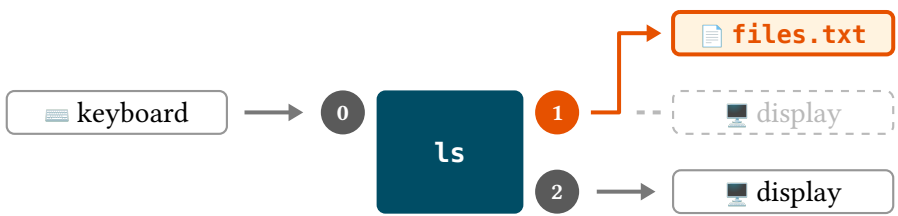
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`>` - redirect `stdout`

The process writes into a file instead of the display

```
$ ls                    # writes on the display
Movies  notes.txt  todo.txt
$ ls > files.txt        # nothing shows up
$ ls /nope > files.txt
ls: cannot access '/nope': No such file or directory
```





💡 only `1` (`stdout`) is redirected, errors still go to the display through `2`, use `2>` for them

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`>>` - append to `stdout`

```
$ echo one > log.txt    # empty the file, then write
```

📄 log.txt

```
one
```

	the shell opens the file with	writes go
`>`	`open("log.txt", O_WRONLY | O_CREAT | O_TRUNC)`	after the file is **emptied**
`>>`	`open("log.txt", O_WRONLY | O_CREAT | O_APPEND)`	at the **end**, always

💡 `>` uses `O_TRUNC`: the file is emptied before `echo` writes `one`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`>>` - append to `stdout`

```
$ echo one > log.txt    # empty the file, then write
$ echo two >> log.txt   # write at the end
```

📄 log.txt

```
one
two
```

	the shell opens the file with	writes go
`>`	`open("log.txt", O_WRONLY | O_CREAT | O_TRUNC)`	after the file is **emptied**
`>>`	`open("log.txt", O_WRONLY | O_CREAT | O_APPEND)`	at the **end**, always

💡 `>>` uses `O_APPEND` instead: nothing is deleted, the kernel moves every write to the end of the file

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`>>` - append to `stdout`



```
$ echo one > log.txt    # empty the file, then write
$ echo two >> log.txt   # write at the end
$ cat log.txt
one
two
```

📄 log.txt

one
two

	the shell opens the file with	writes go
`>`	`open("log.txt", O_WRONLY | O_CREAT | O_TRUNC)`	after the file is **emptied**
`>>`	`open("log.txt", O_WRONLY | O_CREAT | O_APPEND)`	at the **end**, always

💡 useful for logs: every command adds its lines, nothing is lost

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

2> - redirect `stderr`

The errors go into a file instead of the display

```
$ ls /nope                # the error is on the display
ls: cannot access '/nope': No such file or directory
```

FD Table of `ls`	
0	⌨️ keyboard
1	🖥️ display
2	🖥️ display



💡 errors go through `2` (`stderr`), to the display

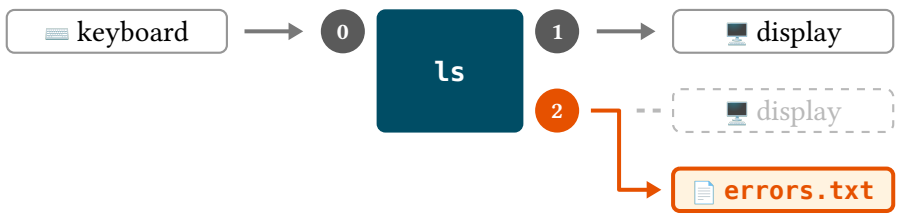
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

2> - redirect stderr

The errors go into a file instead of the display

```
$ ls /nope                # the error is on the display
ls: cannot access '/nope': No such file or directory
$ ls /nope 2> errors.txt  # nothing shows up
```

FD Table of ls	
0	keyboard
1	display
2	errors.txt



💡 `2>` **replaces** `stderr`: `2` is no longer connected to the display, it is connected to `errors.txt`

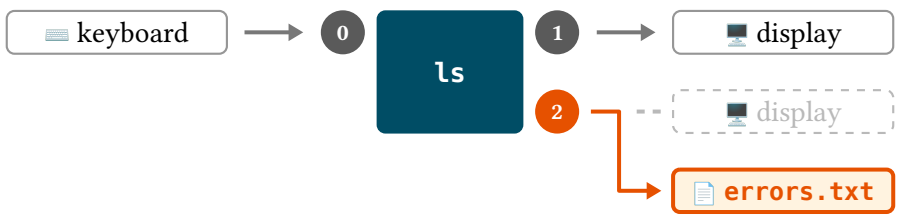
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

2> - redirect stderr

The errors go into a file instead of the display

```
$ ls /nope                # the error is on the display
ls: cannot access '/nope': No such file or directory
$ ls /nope 2> errors.txt  # nothing shows up
$ cat errors.txt
ls: cannot access '/nope': No such file or directory
```

FD Table of `ls`	
0	⌨ keyboard
1	🖥 display
2	📄 errors.txt



💡 `1` is not changed: `> out.txt 2> errors.txt` separates them, `&> all.txt` sends both to the same file

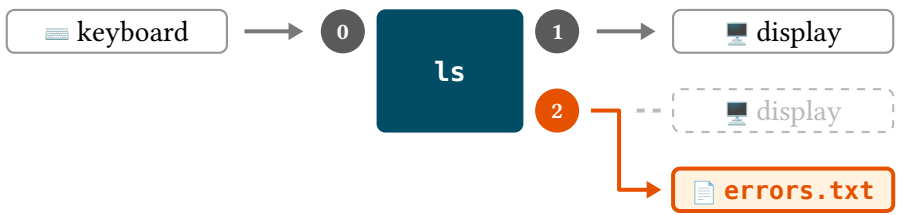
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

2> - redirect stderr

The errors go into a file instead of the display

```
$ ls /nope 2> errors.txt  # the first error
$ ls /nada 2>> errors.txt # append the error
$ cat errors.txt
ls: cannot access '/nope': No such file or directory
ls: cannot access '/nada': No such file or directory
```

FD Table of `ls`	
0	⌨️ keyboard
1	🖥️ display
2	📄 errors.txt



💡 `2>>` **appends**, like `>>`: the first error stays, the new one is added at the end

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`|` - pipe

The `stdout` of a process becomes the `stdin` of the next one



1. the **shell** creates a *pipe*, a buffer in the kernel with a *write* end and a *read* end

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`|` - pipe

The `stdout` of a process becomes the `stdin` of the next one

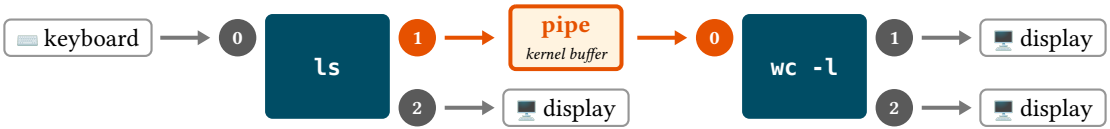


1. the **shell** creates a *pipe*, a buffer in the kernel with a *write* end and a *read* end
2. it starts `ls` and places the *write end* at index `1`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`|` - pipe

The `stdout` of a process becomes the `stdin` of the next one

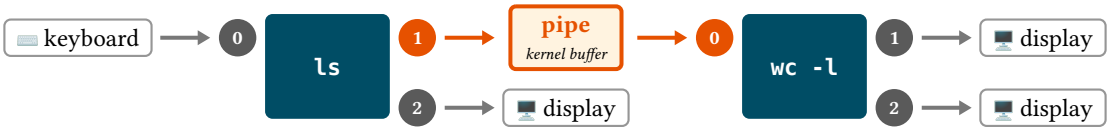


1. the **shell** creates a *pipe*, a buffer in the kernel with a *write* end and a *read* end
2. it starts `ls` and places the *write end* at index `1`
3. it starts `wc` and places the *read end* at index `0`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

| - pipe

The `stdout` of a process becomes the `stdin` of the next one



1. the **shell** creates a *pipe*, a buffer in the kernel with a *write* end and a *read* end
2. it starts `ls` and places the *write end* at index `1`
3. it starts `wc` and places the *read end* at index `0`
4. both run **at the same time**: `wc` reads what `ls` writes, until `ls` ends (*end of file*)

```
$ ls
Movies  notes.txt  todo.txt
$ ls | wc -l
3
```

💡 no temporary file is used

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Files

Devices made by the kernel, useful in redirects

```
/dev
├── null
├── zero
└── urandom
```

🕳 `/dev/null` - the black hole

- everything **written** is *discarded*
- **reading** gives *end of file* right away

```
$ ls /nope 2> /dev/null   # the error is discarded
$ cat < /dev/null         # nothing to read
```



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Files

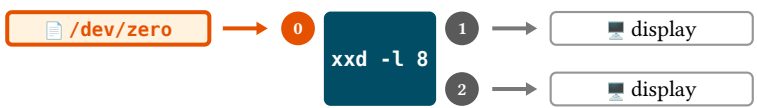
Devices made by the kernel, useful in redirects

```
/dev
├── null
├── zero
└── urandom
```

0️⃣ `/dev/zero` - endless zeros

- **reading** gives as many `0` bytes as asked
- everything **written** is *discarded*

```
$ xxd -l 8 < /dev/zero    # show the first 8 bytes
00000000: 0000 0000 0000 0000                      ........
```



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Files

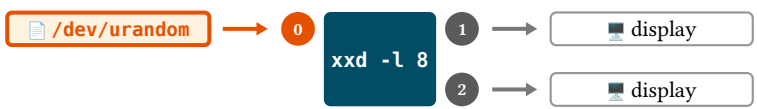
Devices made by the kernel, useful in redirects

```
/dev
├── null
├── zero
└── urandom
```

🎲 `/dev/urandom` - endless random bytes

- **reading** gives as many *random* bytes as asked
- used for keys, passwords, random file names

```
$ xxd -l 8 < /dev/urandom
00000000: 9f3a 51c2 07e8 b4d1                      .:Q.....
$ xxd -l 8 < /dev/urandom # different every time
00000000: 4be0 1d77 c3a9 0f62                      K..w...b
```



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Files

Devices made by the kernel, useful in redirects

```
/dev
├── null
├── zero
└── urandom
```

all of them are *character devices*

- `c` instead of `-` (file) or `d` (directory), **not files**: nothing is stored on the disk
- `1, 3` - the *driver* (major `1`) and the device (minor `3`) that answer




```
$ ls -ld /etc/hostname /dev /dev/null /dev/zero /dev/urandom
-rw-r--r--  1 root root    7 Oct  2 09:12 /etc/hostname  ← regular file
drwxr-xr-x 18 root root 3960 Oct  2 09:12 /dev           ← directory
crw-rw-rw-  1 root root 1, 3 Oct  2 09:12 /dev/null      ← character device
crw-rw-rw-  1 root root 1, 5 Oct  2 09:12 /dev/zero      ← character device
crw-rw-rw-  1 root root 1, 9 Oct  2 09:12 /dev/urandom   ← character device
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Redirects Cheat Sheet

Trick processes about the keyboard, display and error display

```
# Redirect stdout (standard output) to a file
ls > files.txt                      # Save the list of files into files.txt

# Append stdout to a file instead of overwriting it
ls >> files.txt                     # Add more output to files.txt

# Redirect stderr (standard error) to a file
ls /nonexistent 2> errors.txt       # Save only the error message to errors.txt

# Redirect both stdout and stderr to the same file
ls /etc /nonexistent > all.txt 2>&1 # Combine normal output and errors into all.txt

# Shortcut for redirecting both stdout and stderr (Bash 4+)
ls /etc /nonexistent &> all.txt     # Equivalent to the previous command

# Redirect stdin (standard input) from a file
sort < input.txt                    # Read lines from input.txt and sort them

# Redirect stdin and stdout at once
sort < unsorted.txt > sorted.txt    # Sort contents of unsorted.txt into sorted.txt
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Advanced Redirects

Trick processes about the keyboard, display and error display

```
# Suppress all output (stdout and stderr)
command > /dev/null 2>&1            # Run silently; discard all output and errors

# Redirect stderr to stdout and pipe to another command
command 2>&1 | grep "warning"       # Merge both streams, then search for "warning"

# Use a heredoc (redirect stdin from inline text)
cat <<EOF                           # 'cat' reads the next lines, up to EOF
Hello, world!
This is a heredoc example.
EOF
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Advanced Redirects

Trick processes about the keyboard, display and error display

```
# Suppress all output (stdout and stderr)
command > /dev/null 2>&1            # Run silently; discard all output and errors

# Redirect stderr to stdout and pipe to another command
command 2>&1 | grep "warning"       # Merge both streams, then search for "warning"

# Use a heredoc (redirect stdin from inline text)
cat <<EOF                           # 'cat' reads the next lines, up to EOF
Hello, world!
This is a heredoc example.
EOF
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Advanced Redirects

Trick processes about the keyboard, display and error display

```
# Suppress all output (stdout and stderr)
command > /dev/null 2>&1            # Run silently; discard all output and errors

# Redirect stderr to stdout and pipe to another command
command 2>&1 | grep "warning"       # Merge both streams, then search for "warning"

# Use a heredoc (redirect stdin from inline text)
cat <<EOF                           # 'cat' reads the next lines, up to EOF
Hello, world!
This is a heredoc example.
EOF
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Advanced Redirects

Trick processes about the keyboard, display and error display

```
# Suppress all output (stdout and stderr)
command > /dev/null 2>&1            # Run silently; discard all output and errors

# Redirect stderr to stdout and pipe to another command
command 2>&1 | grep "warning"       # Merge both streams, then search for "warning"

# Use a heredoc (redirect stdin from inline text)
cat <<EOF                           # 'cat' reads the next lines, up to EOF
Hello, world!
This is a heredoc example.
EOF
```

💡 the lines of a heredoc are sent *as they are* to `stdin`, a `#` there is not a comment

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Advanced Redirects

Trick processes about the keyboard, display and error display

```
# Suppress all output (stdout and stderr)
command > /dev/null 2>&1            # Run silently; discard all output and errors

# Redirect stderr to stdout and pipe to another command
command 2>&1 | grep "warning"       # Merge both streams, then search for "warning"

# Use a heredoc (redirect stdin from inline text)
cat <<EOF                           # 'cat' reads the next lines, up to EOF
Hello, world!
This is a heredoc example.
EOF
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Soft Links

Shortcuts to files and folders

Bibliography

For this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 2 - *Basic Commands and Directory Hierarchy*
     - Section 2.17.2 - *Working with Symbolic Links*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 2 - *Utilizarea sistemului de fișiere*
     - Section 2.3.4.1 - *Crearea legăturilor simbolice*

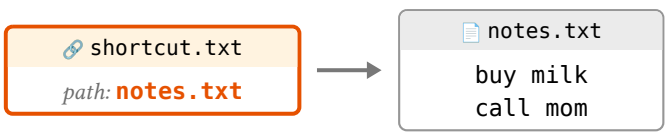
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A Shortcut

A file that points to another file

```
SYNOPSIS
LN(1)  ln [OPTION]... [-T] TARGET LINK_NAME
```

```
$ ln -s notes.txt shortcut.txt   # create the link
```



💡 `ln -s` creates a **soft link** (*symbolic link*)
`LINK_NAME`: a **shortcut** to `TARGET`

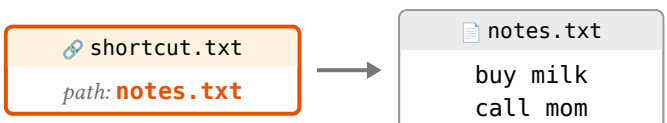
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A Shortcut

A file that points to another file

```
SYNOPSIS
LN(1)  ln [OPTION]... [-T] TARGET LINK_NAME
```

```
$ ln -s notes.txt shortcut.txt   # create the link
$ ls -l
-rw-r--r-- 1 alice alice 18 Oct 12 09:12 notes.txt
lrwxrwxrwx 1 alice alice  9 Oct 12 09:12 shortcut.txt -> notes.txt
```



💡 `l` - the type is *link*, and `ls -l` shows where it points: `-> notes.txt`

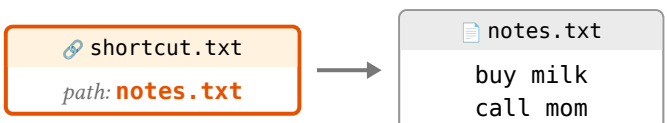
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A Shortcut

A file that points to another file

```
SYNOPSIS
LN(1)  ln [OPTION]... [-T] TARGET LINK_NAME
```

```
$ ln -s notes.txt shortcut.txt   # create the link
$ ls -l
-rw-r--r-- 1 alice alice 18 Oct 12 09:12 notes.txt
lrwxrwxrwx 1 alice alice  9 Oct 12 09:12 shortcut.txt -> notes.txt
$ cat shortcut.txt               # reads notes.txt
buy milk
call mom
```



💡 using the link uses the **target**: `cat` reads the lines of `notes.txt`

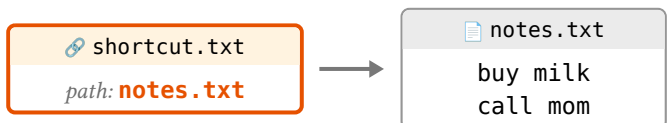
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Contents of a Link

Just a path to a file or a folder

```
$ readlink shortcut.txt
notes.txt
```

- the link does **not** hold the data, it holds a **path**
- the size of `shortcut.txt` is `9`: the length of the text `notes.txt`



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Contents of a Link

Just a path to a file or a folder

```
$ readlink shortcut.txt
notes.txt
$ ln -s /usr/share/doc docs
$ readlink docs
/usr/share/doc
```

- the link does **not** hold the data, it holds a **path**
- the size of `shortcut.txt` is `9`: the length of the text `notes.txt`
- the target can be a **file** or a **folder**
- the path can be *absolute* (`/usr/share/doc`) or *relative* to the folder of the link (`notes.txt`)

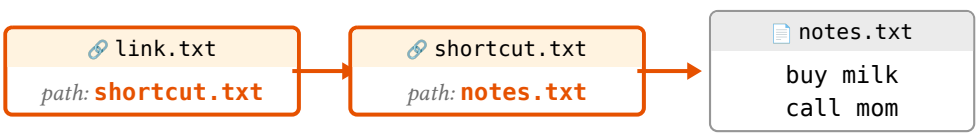


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Following Links

The kernel reads the path and opens the target instead

```
$ ln -s shortcut.txt link.txt    # a link to a link
$ cat link.txt
buy milk
call mom
```



💡 a link can point to **another link**: the kernel follows them all, until it finds a real file

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Following Links

The kernel reads the path and opens the target instead

```
$ ln -s shortcut.txt link.txt    # a link to a link
$ cat link.txt
buy milk
call mom
$ ls docs/                       # into the folder
bash  coreutils  git  ...
```



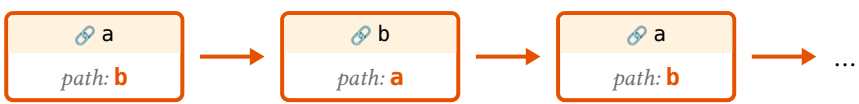
💡 a link to a folder works like the folder: `ls`, `cd`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Following Links

Loops, and commands that do not follow

```
$ ln -s a b                    # b -> a
$ ln -s b a                    # a -> b, a loop
$ cat a
cat: a: Too many levels of symbolic links
```



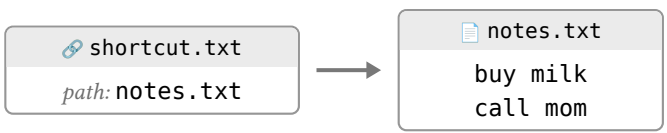
💡 a loop never gets to a real file: the kernel gives up after a few links

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Following Links

Loops, and commands that do not follow

```
$ ln -s a b                          # b -> a
$ ln -s b a                          # a -> b, a loop
$ cat a
cat: a: Too many levels of symbolic links
$ rm link.txt                        # deletes only the link
$ cat shortcut.txt
buy milk
```



⚠ `ls -l`, `readlink` and `rm` do **not** follow: they work on the **link itself**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Broken Links

The target is gone, the link stays

```
$ rm notes.txt                    # delete the target
```



💡 deleting the target does **not** delete the links to it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Broken Links

The target is gone, the link stays

```
$ rm notes.txt                    # delete the target
$ ls -l shortcut.txt
lrwxrwxrwx 1 alice alice  9 Oct 12 09:12 shortcut.txt -> notes.txt
```



💡 the link still holds its path, but `notes.txt` does not exist: the link is **broken**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Broken Links

The target is gone, the link stays

```
$ rm notes.txt                    # delete the target
$ ls -l shortcut.txt
lrwxrwxrwx 1 alice alice  9 Oct 12 09:12 shortcut.txt -> notes.txt
$ cat shortcut.txt
cat: shortcut.txt: No such file or directory
```



- 💡 following a broken link fails
- ⚠️ `ln -s` does not check that the target exists

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Relative Paths

They start from the folder **of the link**, not from the folder you are in

```
$ cd /home/alice
$ ln -s notes.txt backup/wrong.txt     # from backup/
$ cat backup/wrong.txt
cat: backup/wrong.txt: No such file or directory
```

```
/home/alice
├── notes.txt
└── backup
    └── wrong.txt
```



💡 `wrong.txt` is in `backup/`, so the kernel looks for `notes.txt` **in** `backup/`: the link is broken from the start

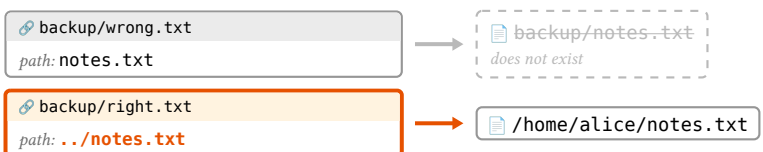
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Relative Paths

They start from the folder **of the link**, not from the folder you are in

```
$ cd /home/alice
$ ln -s notes.txt backup/wrong.txt     # from backup/
$ cat backup/wrong.txt
cat: backup/wrong.txt: No such file or directory
$ ln -s ../notes.txt backup/right.txt  # go up
$ cat backup/right.txt
buy milk
```

```
/home/alice
├── notes.txt
└── backup
    ├── wrong.txt
    └── right.txt
```



💡 `../notes.txt` goes up from `backup/` to `/home/alice`: the file is found

⚠️ write a relative path as if you were in the **folder of the link**

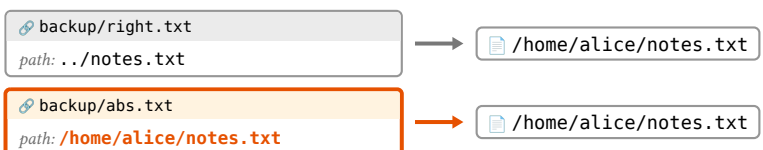
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Moving a Link

The path stays the same, the folder it starts from changes

```
$ ln -s /home/alice/notes.txt backup/abs.txt
```

```
/home/alice
├── notes.txt
└── backup
    ├── right.txt
    └── abs.txt
```



💡 two links in `backup/` to the same file: one with a *relative* path, one with an *absolute* path

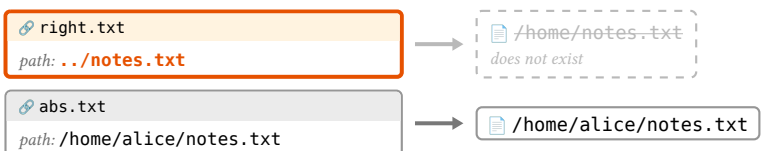
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Moving a Link

The path stays the same, the folder it starts from changes

```
$ ln -s /home/alice/notes.txt backup/abs.txt
$ mv backup/right.txt backup/abs.txt .
$ cat right.txt
cat: right.txt: No such file or directory
```

```
/home/alice
├── notes.txt
├── right.txt  <- moved here
├── abs.txt  <- moved here
└── backup
    ├── right.txt
    └── abs.txt
```



💡 `mv` moves both links up; `right.txt` still holds `../notes.txt`, but now it starts from `/home/alice`: the link is **broken**

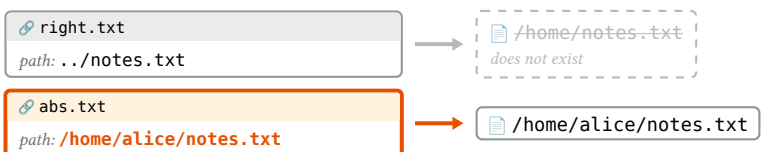
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Moving a Link

The path stays the same, the folder it starts from changes

```
$ ln -s /home/alice/notes.txt backup/abs.txt
$ mv backup/right.txt backup/abs.txt .
$ cat right.txt
cat: right.txt: No such file or directory
$ cat abs.txt
buy milk
call mom
```

```
/home/alice
├── notes.txt
├── right.txt  <- moved here
├── abs.txt  <- moved here
└── backup
    ├── right.txt
    └── abs.txt
```



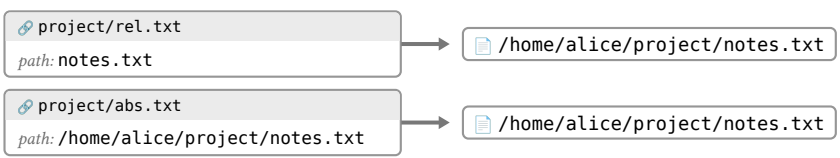
💡 an *absolute* path starts from `/`: it does not matter where the link is

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Moving the Folder

The link and its target move together

```
/
├── home/alice
│   └── project
│       ├── notes.txt
│       ├── rel.txt
│       └── abs.txt
└── tmp
```



💡 `project/` holds a file and two links to it

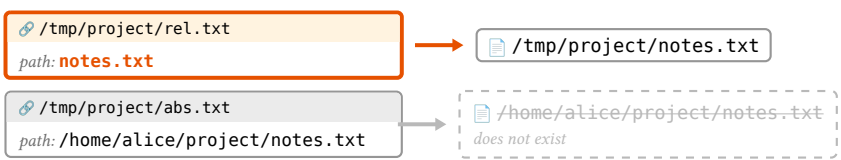
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Moving the Folder

The link and its target move together

```
$ mv project /tmp/project      # the whole folder
$ cat /tmp/project/rel.txt
buy milk
call mom
```

```
/
├── home/alice
│   └── ~~project~~
└── tmp
    └── project  <- moved here
        ├── notes.txt
        ├── rel.txt
        └── abs.txt
```



💡 `rel.txt` starts from its own folder, which moved together with `notes.txt`: it still works

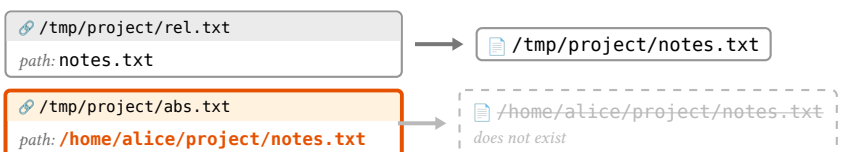
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Moving the Folder

The link and its target move together

```
$ mv project /tmp/project     # the whole folder
$ cat /tmp/project/rel.txt
buy milk
call mom
$ cat /tmp/project/abs.txt
cat: /tmp/project/abs.txt: No such file or directory
```

```
/
├── home/alice
│   └── ~~project~~
└── tmp
    └── project  <- moved here
        ├── notes.txt
        ├── rel.txt
        └── abs.txt
```



💡 `abs.txt` still points to the old place of the file: the link is **broken**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Relative or Absolute?

Choose the path by what will move

	relative path	absolute path
the path starts from	the folder of the link	`/`
only the **link** is moved	**broken**	works
the link and the target are moved **together**	works	**broken**
the **target** is moved or deleted	**broken**	**broken**

💡 *relative* links for files that travel together (a project, an archive), *absolute* links for files that stay in place (`/usr/share/doc`, `/etc/...`)

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The `/proc` Folder

Look inside the processes

Bibliography

For this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 4 - *Disks and Filesystems*
     - Section 4.2.12 - *Special-Purpose Filesystems*
   - Chapter 8 - *A Closer Look at Processes and Resource Utilization*
     - Section 8.1 - *Tracking Processes*
     - Section 8.2 - *Finding Open Files with lsof*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 4 - *Procese*
     - Section 4.2.2 - *Utilitare pentru urmărirea proceselor*
     - Section 4.8 - *Anexa: Sistemul de fișiere procfs*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc` - the process file system

Files made by the kernel, when they are read

```
/proc
├── 1          # systemd
├── 2          # kthreadd
├── ...
├── 4100       # bash
├── 4321       # ...
├── self
├── cpuinfo
├── meminfo
├── uptime
└── ...
```

- `/proc` is a *virtual* file system: every process has a folder, its name is the **PID**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc` - the process file system

Files made by the kernel, when they are read

```
/proc
├── 1          # systemd
├── 2          # kthreadd
├── ...
├── 4100       # bash
├── 4321       # ...
├── self
├── cpuinfo
├── meminfo
├── uptime
└── ...
```

```
$ ls -l /proc/uptime    # the size is 0
-r--r--r-- 1 root root 0 Oct 12 09:12 /proc/uptime
$ cat /proc/uptime      # but it has contents
244699.78 368114.32
```

- `/proc` is a *virtual* file system: every process has a folder, its name is the **PID**
- **nothing** is stored on the disk: the **kernel** writes the contents when a process **reads** the file

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc` - the process file system

Files made by the kernel, when they are read

```
/proc
├── 1          # systemd
├── 2          # kthreadd
├── ...
├── 4100       # bash
├── 4321       # ...
├── self
├── cpuinfo
├── meminfo
├── uptime
└── ...
```

```
$ ls -l /proc/uptime    # the size is 0
-r--r--r-- 1 root root 0 Oct 12 09:12 /proc/uptime
$ cat /proc/uptime      # but it has contents
244699.78 368114.32
$ readlink /proc/self   # the PID of readlink
4325
```

- `/proc` is a *virtual* file system: every process has a folder, its name is the **PID**
- **nothing** is stored on the disk: the **kernel** writes the contents when a process **reads** the file
- `/proc/self` - the folder of the process that reads it: `ls /proc/self` lists the folder of `ls`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>` - inside a process

The resources of a process, as files

```
/proc/4321
├── cmdline   # the arguments
├── comm      # the name
├── cwd       # -> the current folder
├── environ   # the environment variables
├── exe       # -> the executable file
├── fd/       # the File Descriptor Table
├── maps      # the memory map
└── status    # PID, PPID, state, ...
```

Identification

- `status` has the *PID*, the *PPID* and the *state* of the process

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>` - inside a process

The resources of a process, as files

```
/proc/4321
├── cmdline   # the arguments
├── comm      # the name
├── cwd       # -> the current folder
├── environ   # the environment variables
├── exe       # -> the executable file
├── fd/       # the File Descriptor Table
├── maps      # the memory map
└── status    # PID, PPID, state, ...
```

The program

- `exe` - a link to the executable that was loaded (`exec`)
- `cmdline` - the arguments it got
- `comm` - its name

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>` - inside a process

The resources of a process, as files

```
/proc/4321
├── cmdline    # the arguments
├── comm       # the name
├── cwd        # -> the current folder
├── environ    # the environment variables
├── exe        # -> the executable file
├── fd/        # the File Descriptor Table
├── maps       # the memory map
└── status     # PID, PPID, state, ...
```

💾 Data

- `fd/` - one link for every open file descriptor
- `cwd` - a link to the current folder

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>` - inside a process

The resources of a process, as files

```
/proc/4321
├── cmdline   # the arguments
├── comm      # the name
├── cwd       # -> the current folder
├── environ   # the environment variables
├── exe       # -> the executable file
├── fd/       # the File Descriptor Table
├── maps      # the memory map
└── status    # PID, PPID, state, ...
```

🧠 Memory

- `maps` - where the code, the libraries and the variables are in memory
- `environ` - the environment variables

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>` - inside a process

The resources of a process, as files

```
/proc/4321
├── cmdline   # the arguments
├── comm      # the name
├── cwd       # -> the current folder
├── environ   # the environment variables
├── exe       # -> the executable file
├── fd/       # the File Descriptor Table
├── maps      # the memory map
└── status    # PID, PPID, state, ...
```

```
$ readlink /proc/self/exe
/usr/bin/readlink
$ cd /tmp
$ readlink /proc/self/cwd
/tmp
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>/status`

What the kernel knows about a process

```
$ cat /proc/self/status
Name:     cat
Umask:    0022
State:    R (running)
...
Pid:      4330
PPid:     4100
...
Threads:  1
...
```

- `State` - one of the *process states*
- `Threads` - how many threads run in parallel

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/proc/<PID>/status

What the kernel knows about a process

```
$ cat /proc/self/status
Name:     cat
Umask:    0022
State:    R (running)
...
Pid:      4330
PPid:     4100
...
Threads:  1
...
$ echo $$     # the PID of the shell
4100
```

- `State` - one of the *process states*
- `Threads` - how many threads run in parallel
- the `PPid` of `cat` is the shell: the shell called `fork` and `exec` to start it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>/status`

What the kernel knows about a process

```
$ cat /proc/self/status
Name:     cat
Umask:    0022
State:    R (running)
...
Pid:      4330
PPid:     4100
...
Threads:  1
...
$ echo $$     # the PID of the shell
4100
```

- `State` - one of the *process states*
- `Threads` - how many threads run in parallel
- the `PPid` of `cat` is the shell: the shell called `fork` and `exec` to start it

💡 `ps`, `top` and `pstree` do not ask the kernel directly: they **read the files** in `/proc`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>/fd` - the File Descriptor Table

Every open file descriptor is a link

```
$ ls -l /proc/self/fd
total 0
lrwx------ 1 alice alice 64 Oct 12 09:12 0 -> /dev/pts/0
lrwx------ 1 alice alice 64 Oct 12 09:12 1 -> /dev/pts/0
lrwx------ 1 alice alice 64 Oct 12 09:12 2 -> /dev/pts/0
```

FD Table of `ls`

0	/dev/pts/0
1	/dev/pts/0
2	/dev/pts/0

💡 `0`, `1` and `2` point to `/dev/pts/0`, the *terminal*: it is both the keyboard and the display

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>/fd` - the File Descriptor Table

Every open file descriptor is a link

```
$ ls -l /proc/self/fd
total 0
lrwx------ 1 alice alice 64 Oct 12 09:12 0 -> /dev/pts/0
lrwx------ 1 alice alice 64 Oct 12 09:12 1 -> /dev/pts/0
lrwx------ 1 alice alice 64 Oct 12 09:12 2 -> /dev/pts/0
lr-x------ 1 alice alice 64 Oct 12 09:12 3 -> /proc/4331/fd
```

FD Table of `ls`

0	/dev/pts/0
1	/dev/pts/0
2	/dev/pts/0
3	**/proc/4331/fd**

💡 `ls` opened the folder it lists, at the first free index: `3`, just like `sort` opened `fruits.txt`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc/<PID>/fd` - the File Descriptor Table

Every open file descriptor is a link

```
$ lsof -p 4100     # the open files of the shell (PID 4100)
COMMAND  PID USER   FD  TYPE DEVICE SIZE/OFF NODE NAME
...
bash    4100 alice  0u  CHR  136,0      0t0    3 /dev/pts/0
bash    4100 alice  1u  CHR  136,0      0t0    3 /dev/pts/0
bash    4100 alice  2u  CHR  136,0      0t0    3 /dev/pts/0
```

💡 `lsof -p <PID>` lists the same information: it reads `/proc/<PID>/fd`

⚠️ only the **owner** of the process (or `root`) can look inside its `fd/`

💡 a redirect changes these links: with `> out.txt`, `1` points to `out.txt`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc` - the whole system

Not only processes

```
/proc
├── cpuinfo   # the processors
├── meminfo   # the memory
├── uptime    # seconds since boot
├── loadavg   # how busy the system is
└── mounts    # the mounted file systems
```

file	shown by
`/proc/cpuinfo`	`lscpu`
`/proc/meminfo`	`free`
`/proc/uptime`, `/proc/loadavg`	`uptime`
`/proc/mounts`	`mount`

```
$ head -3 /proc/meminfo
MemTotal:       65296068 kB
MemFree:        10651832 kB
MemAvailable:   53764732 kB
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc` - the whole system

Not only processes

```
/proc
├── cpuinfo   # the processors
├── meminfo   # the memory
├── uptime    # seconds since boot
├── loadavg   # how busy the system is
└── mounts    # the mounted file systems
```

file	shown by
`/proc/cpuinfo`	`lscpu`
`/proc/meminfo`	`free`
`/proc/uptime`, `/proc/loadavg`	`uptime`
`/proc/mounts`	`mount`

```
$ head -3 /proc/meminfo
MemTotal:       65296068 kB
MemFree:        10651832 kB
MemAvailable:   53764732 kB
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc` - the whole system

Not only processes

```
/proc
├── cpuinfo   # the processors
├── meminfo   # the memory
├── uptime    # seconds since boot
├── loadavg   # how busy the system is
└── mounts    # the mounted file systems
```

file	shown by
`/proc/cpuinfo`	`lscpu`
`/proc/meminfo`	`free`
`/proc/uptime`, `/proc/loadavg`	`uptime`
`/proc/mounts`	`mount`

```
$ head -3 /proc/meminfo
MemTotal:       65296068 kB
MemFree:        10651832 kB
MemAvailable:   53764732 kB
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc` - the whole system

Not only processes

```
/proc
├── cpuinfo   # the processors
├── meminfo   # the memory
├── uptime    # seconds since boot
├── loadavg   # how busy the system is
└── mounts    # the mounted file systems
```

file	shown by
`/proc/cpuinfo`	`lscpu`
`/proc/meminfo`	`free`
`/proc/uptime`, `/proc/loadavg`	`uptime`
`/proc/mounts`	`mount`

```
$ head -3 /proc/meminfo
MemTotal:       65296068 kB
MemFree:        10651832 kB
MemAvailable:   53764732 kB
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`/proc` - the whole system

Not only processes

```
/proc
├── cpuinfo   # the processors
├── meminfo   # the memory
├── uptime    # seconds since boot
├── loadavg   # how busy the system is
└── mounts    # the mounted file systems
```

file	shown by
`/proc/cpuinfo`	`lscpu`
`/proc/meminfo`	`free`
`/proc/uptime`, `/proc/loadavg`	`uptime`
`/proc/mounts`	`mount`

💡 most of the system tools just **read** `/proc` and format the result nicely

```
$ head -3 /proc/meminfo
MemTotal:       65296068 kB
MemFree:        10651832 kB
MemAvailable:   53764732 kB
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

We talked about

- Installing applications
- Process
- PID
- Process Loading
- Process States
- `fork` and `exec`
- How Linux starts
- Orphan and zombie processes
- File descriptors
- Redirects
- Soft links
- The `/proc` folder

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0